Exploiting symmetries for testing equivalence verification in the SPI calculus

International Journal of Foundations of Computer Science

EXPLOITING SYMMETRIES FOR TESTING EQUIVALENCE VERIFICATION IN THE SPI CALCULUS

IVAN CIBRARIO B., LUCA DURANTE
IEIIT-CNR, C.so Duca degli Abruzzi 24
Torino, I-10129, Italy

RICCARDO SISTO
DAUIN, Politecnico di Torino, C.so Duca degli Abruzzi 24
Torino, I-10129, Italy

and

ADRIANO VALENZANO
IEIIT-CNR, C.so Duca degli Abruzzi 24
Torino, I-10129, Italy

Received (received date)
Revised (revised date)
Communicated by Editor's name

ABSTRACT

Testing equivalence is a quite powerful way of expressing security properties of cryptographic protocols, but its formal verification is a difficult task, because it is based on universal quantification over contexts. A technique based on state exploration to address this verification problem has previously been presented; it relies on an environment-sensitive labelled transition system (ES-LTS) and on symbolic term representation. This paper shows that such a technique can be enhanced by exploiting symmetries found in the ES-LTS structure. Experimental results show that the proposed enhancement can substantially reduce the size of the ES-LTS and that the technique as a whole compares favorably with respect to related work.

Keywords: Spi calculus, Cryptographic protocols, Model checking, Testing equivalence.

1. Introduction

Due to the increasing importance of secure distributed applications the formal verification of cryptographic protocols is being studied extensively by several researchers, through the investigation of proof techniques, based on various proof systems and description formalisms [1, 4, 19, 20], and on state exploration methods [8, 11, 13, 14, 15, 16]. The latter requires modelling the protocol behavior as a reasonably-sized finite state system, which generally entails the introduction of simplifying assumptions and numerical bounds that can reduce the accuracy of

the analysis. Nevertheless, this kind of verification has the invaluable advantage of being fully automatic.

This paper is focused on the spi calculus [2], a process algebra derived from the π-calculus [17] with some simplifications and the addition of cryptographic operations. The testing equivalence formulation of security properties introduced in [2] is more accurate than alternative formulations based on the intruder knowledge [9]; however, the efficient verification of testing equivalence is not trivial because of the need for universal quantification over testers: testing equivalence means that two processes are indistinguishable for *any* tester process, and there are an infinite number of such processes.

This problem was addressed initially in [1] and [4], where tractable proof methods were introduced. Instead, [11] defined a spi calculus dialect and its BRUTUS logic [7], aiming at the definition of a theoretical framework for model checking a set of logic properties, which is possibly wider than the set of those expressed by testing equivalence, on the spi calculus.

More recently, a method for checking the spi calculus testing equivalence using exhaustive state exploration instead of theorem proving was presented in [9]; there, the problem of the quantification over contexts is solved in a way similar to the one reported in [4], i.e. by defining an Environment-Sensitive Labelled Transition System (ES-LTS), which describes the possible evolutions of both the protocol principals and the corresponding intruder knowledge. In [9] it has been shown that trace equivalence defined on such an ES-LTS is a necessary and sufficient condition for testing equivalence.

In order to have a finite model which can be analyzed by state exploration, only spi calculus descriptions which have a finite number of processes are considered, thus ruling out the infinite replication operator of the spi calculus, and symbolic techniques are used to get a finite representation of the infinite set of data that the intruder can send each time a protocol principal performs an input.

To further reduce the model size and keep it within reasonable bounds, this paper introduces a symmetry-based reduction method, which cuts off duplicated behaviors that can be identified by inspecting the state behavior expression. When multiple, concurrent sessions of the same protocol are involved, as often happens in practice, such symmetries show up extensively.

The feasibility of the proposed method and the increase in performance achieved by symmetry-based reductions are shown by the application of a preliminary version of an automatic verification tool, called S^3A (**S**ymbolic **S**pi calculus **S**pecifications **A**nalyser).

This paper, which assumes that the reader is familiar with basic cryptographic techniques and the spi calculus, is organized as follows: Sects. 2 and 3 recall the spi calculus language and the ES-LTS model, respectively. The theory of our symmetry-based reduction technique is presented in Sect. 4 and Sect. 5 offers an example. Sect. 6 gives some experimental results and comparisons, while Sect. 7 concludes the paper and discusses possible further developments.

Table 1: Syntax of the spi calculus.

$\sigma, \rho, \theta ::=$ terms		$P, Q, R ::=$	processes
m	constant	$\overline{\sigma}\langle\rho\rangle.P$	output
0	the zero constant	$\sigma(x).P$	input
x, y	variables	$P \mid Q$	composition
(σ, ρ)	pair	$(\nu\ m)\ P$	restriction
$suc(\sigma)$	successor	0	nil
$H(\sigma)$	hashing	$[\sigma\ is\ \rho]\ P$	match
$\{\sigma\}_\rho$	shared-key encr.	$let\ (x, y)\ =\ \sigma\ in\ P$	pair splitting
σ^+	public part	$case\ \sigma\ of\ 0 : P\ suc(x) : Q$	integer case
σ^-	private part	$case\ \sigma\ of\ \{x\}_\rho\ in\ P$	shared-key decr.
$\{[\sigma]\}_\rho$	public-key encr.	$case\ \sigma\ of\ \{[x]\}_\rho\ in\ P$	decryption
$[\{\sigma\}]_\rho$	private-key sign.	$case\ \sigma\ of\ [\{x\}]_\rho\ in\ P$	sign. check

2. The Spi Calculus

The syntax of the spi calculus provides two basic language elements: *terms*, which represent data (e.g. messages, channel identifiers, keys, key pairs, integers), and *processes*, which represent behaviors. Terms can be either atomic constants and variables, or structured terms built using term composition operators.

Table 1 outlines the syntax of the spi calculus [2]. The left-hand side of the table shows the term grammar and presents some naming conventions used in this paper. Besides term specification, the spi calculus also provides a rich set of process algebraic operators, which are used to build behavior expressions, and they are summarized on the right-hand side of Table 1.

For example, Fig. 1 presents the spi calculus specification of a simple protocol, inspired by [10]: two agents A and B share a secret key k and exchange two simple messages. The left-hand side of Fig. 1 shows the message exchanges the protocol requires, adopting the informal, intuitive notation that is often used in the literature dealing with security protocols, whereas the right-hand side shows the spi calculus specification of the protocol. Processes P_A and P_B represent the roles of agents A and B, respectively. The process P_{sample} puts the two behaviors in parallel, allowing them to be synchronized on channel c. Since c is public, it can also be accessed by a potential intruder.

First of all, A sends B a message M encoded by means of k over public channel c, then B tries to decode it and, in case of success, acknowledges A by sending it back the hashed cleartext. Finally, A checks that the received message matches with $H(M)$ and then proceeds with further operations. $F(M)$ is an unspecified process; it represents the behavior of A after the successful termination of the protocol session.

Let us now introduce some further notations and definitions about spi calculus, that will be used throughout the paper.

$fv(P)$ and $fc(P)$ denote the set of free variables and constants occurring in process P, respectively. Both sets can be computed easily by syntactically inspecting process P; for simplicity, it is assumed that name overloading is not allowed. Also,

1) $\mathtt{A} \rightarrow \mathtt{B} : \{M\}_k$ 2) $\mathtt{B} \rightarrow \mathtt{A} : H(M)$	$P_A(M) \triangleq \overline{c}\langle\{M\}_k\rangle.\ c(x).\ [x\ is\ H(M)]\ F(M)$ $P_B \triangleq c(y_1).\ case\ y_1\ of\ \{y_2\}_k\ in\ \overline{c}\langle H(y_2)\rangle.\ 0$ $P_{sample}(M) \triangleq (\nu k)(P_A(M) \mid P_B)$

Figure 1: A sample spi calculus specification.

$\mathcal{A}$ denotes the set of spi calculus names, and $\mathcal{M}(\mathcal{A})$ the set of all spi calculus terms that can be built starting from $\mathcal{A}$.

The usual implicit assumptions about perfect encryption apply in the spi calculus as well as in other similar specification formalisms.

3. The Environment-Sensitive LTS model

The *environment-sensitive labelled transition system* (ES-LTS) defined in [9] describes all possible interactions of a given spi calculus process with its environment. In such a model, each time the spi calculus process executes an input, the environment can send it any data term that can be built starting from the current environment knowledge. Since the set of these terms is infinite, it is represented symbolically by a so-called generic term, so as to have a finite ES-LTS.

Each state of the ES-LTS is denoted $(K \rhd P)_{\Upsilon,\Lambda}$ and is made up of the current spi calculus process P, the current environment knowledge K, and a specification of how symbolic terms occurring in P and K must be interpreted (Υ, Λ). Each ES-LTS transition takes the general form:

$$(K \rhd P)_{\Upsilon,\Lambda} \quad \stackrel{\mu}{\underset{\phi}{\longmapsto}} \quad (K' \rhd P')_{\Upsilon',\Lambda'} \ , \tag{1}$$

where μ and ϕ are synthetic representations of the action performed by process P and of the complementary action performed by the environment, respectively.

3.1. Knowledge Representation

The environment knowledge is represented in a minimized and canonical form. It includes a set of messages learned by the intruder, with a labeling on its elements that uniquely identifies them according to the order in which they have been added to the set. The need for such a labeling derives from the features of testing equivalence. If the interest was only in checking simple security properties related to what data the intruder can generate, no labeling would be needed. If instead interest lies in verifying testing equivalence, the intruder's ability to classify the data items of its knowledge has to be modeled according to when they have been learned. For example, the spi calculus processes $P \triangleq \overline{c}\langle M\rangle.\overline{c}\langle N\rangle.\overline{c}\langle M\rangle.0$ and $Q \triangleq \overline{c}\langle M\rangle.\overline{c}\langle N\rangle.\overline{c}\langle N\rangle.0$ are not testing equivalent, although the set of data that the intruder can learn from each of them is the same at each step. In fact, a simple test that can distinguish between P and Q is the one that checks whether the third received data is equal to the first one.

The labeling on the environment knowledge data items solves this problem and, at the same time, makes it possible to identify such data items independently of their

particular value, which is another key feature needed in checking testing equivalence. For example, the spi processes $P \triangleq (\nu k)\overline{c}\langle\{M\}_k\rangle.0$ and $Q \triangleq (\nu k)\overline{c}\langle\{N\}_k\rangle.0$ are testing equivalent although their outputs are different; this can readily be recognized by comparing the identifiers that the intruder assigns to the learned data, instead of comparing the data themselves.

The environment knowledge is formally represented by a bijective function $K : \overline{\Sigma} \rightarrow L$, where the domain $\overline{\Sigma}$ is the set of terms that the intruder has learned, and the image L is the finite set of indexes uniquely identifying them. The countable, ordered set of indexes is denoted as I, so that $L \subseteq I$. Of course, the intruder term generation capabilities depend on $\overline{\Sigma}$. If $\sigma \in \mathcal{M}(\mathcal{A})$ is a finite term, it is said that σ can be *produced* by K, written $K \vdash \sigma$, iff σ belongs to the closure of $\overline{\Sigma}$ with respect to the spi calculus term composition operators, i.e. $\sigma \in \widehat{\overline{\Sigma}}$.

Similarly, it is said that σ can be produced by $\overline{\Sigma}$, written $\overline{\Sigma} \vdash \sigma$, iff $\sigma \in \widehat{\overline{\Sigma}}$. As shown in [9], $\overline{\Sigma}$ is always kept in a minimized and canonical form. The decidability of $K \vdash \sigma$ and of $\overline{\Sigma} \vdash \sigma$ has been proved in [9] and in [6], where the algorithms to check them and to incrementally compute $\overline{\Sigma}$ are given. In the following, $K' = f(\rho, K)$ denotes the new knowledge that is reached from K after the environment has observed the data term ρ.

3.2. Symbolic Data Representation

As already noted, an input action of the current spi calculus process is represented in the ES-LTS by means of a generic data term, which represents the infinite set of terms that the environment can generate at that moment. Generic terms are added to the spi calculus by extending the language with an additional infinite and countable set of names Γ, such that $\Gamma \cap \mathcal{A} = \emptyset$ and $\Gamma \cap I = \emptyset$. In the rest of this paper, γ ranges over Γ. A generic term is a spi calculus term $\theta \in \mathcal{M}(\mathcal{A} \cup I \cup \Gamma)$ which has at least one subterm $\gamma \in \Gamma$.

Each symbolic ES-LTS state is characterized by a (finite) current set of atomic generic terms $G \subset \Gamma$. A function $\Upsilon : G \rightarrow 2^{\mathcal{M}(\mathcal{A} \cup I \cup G)}$, which is part of the symbolic ES-LTS state, gives the current interpretation of atomic generic terms. Each $\gamma \in G$ is mapped onto a corresponding knowledge function domain $\Upsilon(\gamma)$ that represents the set of terms that was available to the intruder when γ was generated.

When the term represented by γ is generated by the intruder it can take the whole set of values $\widehat{\Upsilon(\gamma)}$, and a different subsequent behavior is possible for each one of them. However, for the purpose of testing equivalence, such behaviors are indistinguishable from one another, and can be represented as a single symbolic behavior, until the occurrence of some operation is conditioned by the value that was initially exchanged. Whenever this happens, only the behaviors corresponding to values that satisfy the condition are allowed to proceed, and this is symbolically described by narrowing the set of terms represented by each atomic generic term γ down to the largest subset of $\widehat{\Upsilon(\gamma)}$ compatible with the operations performed.

In most cases, a narrowing of this kind is equivalent to the substitution of one or more atomic generic terms with corresponding *specialized* generic terms or concrete

terms. A specialized generic term is a compound term which has at least one atomic generic term as a subterm. For example, a pair splitting operation on a generic term γ narrows the set of concrete terms represented by γ so as to include only pairs of elements of $\widehat{\Upsilon(\gamma)}$, which is equivalent to applying the substitution $\langle(\gamma', \gamma'')/\gamma\rangle$, where γ' and γ'' are two new atomic generic terms representing the two components of the pair.

Substitutions like the ones presented above are called *specializations*, because they substitute atomic generic terms with corresponding specialized or concrete ones. For this reason, atomic generic terms are also called unspecialized generic terms. In the rest of this paper, ξ ranges over specializations.

Specializations are not powerful enough to precisely describe any kind of narrowing that can occur on the sets of terms represented by generic terms. For this reason, *extended narrowing specifications* are introduced, which makes it possible to specify that a given specialization must occur, while one or more further specializations can not occur. Such narrowing specializations take the form of a pair $\langle \xi, \delta_\Lambda \rangle$ where ξ is the specialization that must be applied and $\delta_\Lambda = \{\xi_1, \ldots \xi_k\}$ is the set of forbidden further specializations. Of course, $\langle \xi, \emptyset \rangle = \xi$.

As long as the computation proceeds, forbidden specializations are added up, and the set of specializations obtained as the union of all forbidden specializations accumulated up to the current state is denoted Λ. It complements Υ in specifying the current interpretation of generic terms.

The set of all the specializations that can be applied in the current state depends only on Υ and Λ and is denoted by $S_{\Upsilon,\Lambda}$. When a specialization ξ is applied to a symbolic state, Υ and Λ are updated accordingly, and the new interpretation is denoted by $\Upsilon\{\xi\}, \Lambda\{\xi\}$.

3.3. Canonical Representation

As already mentioned, when checking testing equivalence it is necessary to abstract away from the exact value of the exchanged data. The only relevant thing is how such data are related to the current intruder knowledge. The *canonical representation of a term σ with respect to an intruder knowledge K* expresses how σ is related to K. This concept can be introduced by extending the notion of substitution. The substitutions originally introduced in [2] act on atomic terms only. If this constraint is relaxed, it is possible to have substitution lists $\lambda = \sigma_1/\rho_1, \sigma_2/\rho_2, \ldots, \sigma_n/\rho_n$, where some ρ_i are non-atomic terms. If λ is one of these extended substitutions, the postfix operator $[\lambda]$ replaces each occurrence of term ρ_i with term σ_i.

The canonical representation of a term σ with respect to an intruder knowledge K is a spi calculus term defined over the extended set of names $\mathcal{A} \cup \Gamma \cup I$, obtained by applying to σ a substitution that replaces each subterm $\rho \in Dom(K)$ by its corresponding unique identifier $K(\rho)$. Such a substitution is represented by a substitution list made up of an item $K(\rho)/\rho$ for each $\rho \in Dom(K)$. With abuse of notation, this substitution is denoted K, and, consequently, the canonical representation is written $\sigma[K]$.

Since $[K]$ substitutes each occurrence of ρ with its corresponding index $K(\rho)$, $\sigma[K]$ actually specifies how σ can be built using the data items available in the intruder knowledge, each one being identified by its index. If $K \vdash \sigma$, then $\sigma[K] \in \mathcal{M}(\Gamma \cup I)$, i.e. the canonical representation of a term that can be produced by K does not contain any spi calculus name, but only generic terms and indexes, because it can be built using the elements of the intruder knowledge only. In a similar way, it is possible to define the canonical representation of any object containing spi calculus terms.

3.4. The ES-LTS Derivation System

Transitions can be categorized into three different types, taking the following forms:

$$(K \rhd P)_{\Upsilon,\Lambda} \quad \xrightarrow[\xi[K']]{\tau} \quad (K' \rhd P')_{\Upsilon',\Lambda'} \tag{2}$$

$$(K \rhd P)_{\Upsilon,\Lambda} \quad \xrightarrow[\langle \xi,\delta_\Lambda \rangle [K'],\delta_K]{\overline{\sigma[\xi][K']}} \quad (K' \rhd P')_{\Upsilon',\Lambda'} \tag{3}$$

$$(K \rhd P)_{\Upsilon,\Lambda} \quad \xrightarrow[\gamma]{\sigma[K]} \quad (K \rhd P')_{\Upsilon',\Lambda} \tag{4}$$

All labels (including the component denoted as δ_K) are canonical representations with respect to the new intruder knowledge K'.

Transitions taking form (2) are related to synchronization events occurring inside the spi process. In this case, the process action is denoted by the special symbol τ, which represents an internal synchronization. The complementary action label may contain a pure specialization; in this case the transition is referred to as a *specialization transition*.

A transition taking form (3) is referred to as an *output transition* and represents an output on channel σ. The complementary action label includes an item, denoted by δ_K, which describes how the intruder knowledge is affected by the event, and may contain an extended narrowing specification $\langle \xi, \delta_\Lambda \rangle$. The process action specifies the channel name σ after the application of specialization ξ. The overline symbol indicates, as in the spi calculus, that the operation performed by the process on the channel is an output.

A transition taking form (4) is referred to as an *input transition* and represents an input from channel σ. It implies a data transfer from the environment to the process. Thus, no modification in the environment knowledge takes place. The process action is analogous to the previous one, whereas the complementary action label is a new generic term symbolically representing any data term that can be generated by the intruder.

The ES-LTS derivation system is an extension of the derivation system defined in [2] for the reaction relation. The main rules specify when input and output transitions may take place:

$$\frac{K \vdash \sigma \qquad \langle \xi,\delta_\Lambda \rangle \in \Theta(\rho, K_{\Upsilon,\Lambda}) \qquad K'_{\Upsilon',\Lambda'} = f_{\langle \xi,\delta_\Lambda \rangle}(\rho, K_{\Upsilon,\Lambda})}{(K \rhd \overline{\sigma}\langle \rho \rangle.P)_{\Upsilon,\Lambda} \xrightarrow[\langle \xi,\delta_\Lambda \rangle [K'], \left\langle \delta_{\overline{K}}(\rho),\, \delta_{\overline{\overline{K}}}(\rho),\, \rho \right\rangle [K']}{\overline{\sigma[\xi][K']}} (K' \rhd P[\xi])_{\Upsilon',\Lambda'}} \tag{5}$$

$$\frac{K \vdash \sigma \qquad \gamma \notin dom(\Upsilon)}{(K \rhd \sigma(x).P)_{\Upsilon,\Lambda} \xrightarrow[\gamma]{\sigma[K]} (K \rhd P[\gamma/x])_{\Upsilon \cup \{(\gamma, dom(K))\},\Lambda}} \tag{6}$$

where:

$$f_{\langle \xi, \delta_\Lambda \rangle}(\rho, K_{\Upsilon,\Lambda}) = f(\rho[\xi], K[\xi])_{\Upsilon\{\xi\},(\Lambda \cup \delta_\Lambda)\{\xi\}} \tag{7}$$

$$\Theta(\rho, K_{\Upsilon,\Lambda}) = \{\langle \xi, \delta_\Lambda \rangle \mid \xi \in S_{\Upsilon,\Lambda}, \delta_\Lambda \in S_\Upsilon, f_{\langle \xi, \delta_\Lambda \rangle}(\rho, K_{\Upsilon,\Lambda}) \in \mathcal{K}_{\Upsilon,\Lambda}\} \tag{8}$$

Function $f_{\langle \xi, \delta_\Lambda \rangle}$ is the symbolic version of function f after the application of the narrowing specified by $\langle \xi, \delta_\Lambda \rangle$. As can be seen in (7), it results from the composition of the knowledge transformation implied by the application of ξ, the addition of δ_Λ to Λ, and the knowledge transformation described by function $f()$. $\Theta(\rho, K_{\Upsilon,\Lambda})$ is the set of narrowings $\langle \xi, \delta_\Lambda \rangle$ that make $f_{\langle \xi, \delta_\Lambda \rangle}(\rho, K_{\Upsilon,\Lambda})$ a valid knowledge function. $\mathcal{K}_{\Upsilon,\Lambda}$ is the set of all valid knowledge functions.

It is worth noting that it is possible that the new knowledge function $K'_{\Upsilon',\Lambda'}$, reached after the output of ρ, depend on how generic terms are specialized. Therefore, there may be more than one possible $K'_{\Upsilon',\Lambda'}$, each one corresponding to a different transition and to a different element of $\Theta(\rho, K_{\Upsilon,\Lambda})$. In rule (5), the δ_K of rule (3) has been expanded as $\langle \delta_K^-(\rho), \delta_K^=(\rho), \rho \rangle [K']$, where:

- ρ is the term the process sent as output on the public channel σ, and that has been observed by the environment,
- $\delta_K^-(\rho)$ is the set of all elements that are eliminated from K in the transformation from K to K' when ρ is received by the environment,
- $\delta_K^=(\rho)$ is the set of terms that become decipherable after ρ has been received by the environment, without being eliminated from the intruder knowledge domain.

The semantics of specialization transitions is described by the following rules:

$$\frac{(K \rhd P)_{\Upsilon,\Lambda} \xrightarrow[\xi[K[\xi]]]{\tau} (K \rhd P')_{\Upsilon,\Lambda\{\xi\}}}{(K \rhd (P|Q))_{\Upsilon,\Lambda} \xrightarrow[\xi[K[\xi]]]{\tau} (K \rhd (P'|Q))_{\Upsilon,\Lambda\{\xi\}}} \tag{9}$$

$$\frac{\xi \in \sigma \bullet \rho \qquad \xi \neq \top}{(K \rhd (\overline{\sigma}\langle\theta\rangle.P \mid \rho(x).Q))_{\Upsilon,\Lambda} \xrightarrow[\xi[K[\xi]]]{\tau} (K \rhd (\overline{\sigma}\langle\theta\rangle.P \mid \rho(x).Q))_{\Upsilon,\Lambda\{\xi\}}} \tag{10}$$

$$\frac{\xi \in \sigma \bullet \rho \qquad \xi \neq \top}{(K \rhd ([\sigma \ is \ \rho]P))_{\Upsilon,\Lambda} \xrightarrow[\xi[K[\xi]]]{\tau} (K \rhd [\sigma \ is \ \rho]P)_{\Upsilon,\Lambda\{\xi\}}} \tag{11}$$

$$\frac{\gamma', \gamma'' \notin dom(\Upsilon)}{(K \rhd (let \ (x,y) = \gamma \ in \ P))_{\Upsilon,\Lambda} \xrightarrow[(\gamma',\gamma'')/\gamma]{\tau} (K \rhd (let \ (x,y) = \gamma \ in \ P))_{\Upsilon,\Lambda\{(\gamma',\gamma'')/\gamma\}}} \tag{12}$$

$$\frac{-}{(K \rhd (case \ \gamma \ of \ 0{:}P \ suc(x){:}Q))_{\Upsilon,\Lambda} \xrightarrow[0/\gamma]{\tau} (K \rhd (case \ \gamma \ of \ 0{:}P \ suc(x){:}Q))_{\Upsilon,\Lambda\{0/\gamma\}}} \tag{13}$$

$$\frac{\gamma' \notin dom(\Upsilon)}{(K \rhd (case\ \gamma\ of\ 0{:}P\ suc(x){:}Q))_{\Upsilon,\Lambda} \xrightarrow[suc(\gamma')/\gamma]{\tau} (K \rhd (case\ \gamma\ of\ 0{:}P\ suc(x){:}Q))_{\Upsilon,\Lambda\{suc(\gamma')/\gamma\}}} \tag{14}$$

$$\frac{\xi \in \theta \circ \rho \qquad \xi \neq \top}{(K \rhd (case\ \theta\ of\ \{x\}_\rho\ in\ P))_{\Upsilon,\Lambda} \xrightarrow[\xi[K[\xi]]]{\tau} (K \rhd (case\ \theta\ of\ \{x\}_\rho\ in\ P))_{\Upsilon,\Lambda\{\xi\}}} \tag{15}$$

$$\frac{\xi \in \eta^{\oplus} \rho \qquad \xi \neq \top}{(K \rhd (case\ \eta\ of\ \{[x]\}_\rho\ in\ P))_{\Upsilon,\Lambda} \xrightarrow[\xi[K[\xi]]]{\tau} (K \rhd (case\ \eta\ of\ \{[x]\}_\rho\ in\ P))_{\Upsilon,\Lambda\{\xi\}}} \tag{16}$$

$$\frac{\xi \in \eta^{\ominus} \rho \qquad \xi \neq \top}{(K \rhd (case\ \eta\ of\ [\{x\}]_\rho\ in\ P))_{\Upsilon,\Lambda} \xrightarrow[\xi[K[\xi]]]{\tau} (K \rhd (case\ \eta\ of\ [\{x\}]_\rho\ in\ P))_{\Upsilon,\Lambda\{\xi\}}} \tag{17}$$

Rule (9) specifies how the parallel composition operator is dealt with. The other rules specify all the situations in which specialization transitions can occur. The operators $\bullet$, $\circ$, $\oplus$, and $\ominus$ are unification operations. Each one of them yields the minimal sets of specializations that, when applied, make some condition true. More precisely, $\sigma \bullet \rho$ is the set of specializations that must be applied to σ and ρ so that they can be matched. Formally, $\sigma \bullet \rho \triangleq \{\xi \in S_{\Upsilon,\Lambda} \mid \sigma[\xi] = \rho[\xi]\}$. Similarly, $\sigma \circ \rho$ yields the specializations that make σ a term encrypted under key ρ, whereas $\oplus$ and $\ominus$ are the public and private key variants of $\circ$.

ES-LTS traces are defined in the usual way as sequences of transition labels. In [9] it is proved that trace equivalence defined on the ES-LTS coincides with testing equivalence.

4. Symmetry-Based Reductions

The ES-LTS generation and the consequent trace comparison suffer from the state-explosion problem. In order to reduce the number of states, several techniques [7, 21] have been proposed and adopted, mainly in the field of reachability analysis. Formerly, symmetry-based techniques have been employed, mainly to check for the existence of deadlocks, in the Petri net community [12]. The approach adopted here is mainly inspired by the pioneering work of [21] and, to the best of our knowledge, this is the first time that a symmetry-based reduction technique has been applied to improve the efficiency of automatic *testing equivalence* verification.

In [21], two kinds of symmetry are analyzed and exploited: *process* and *state symmetry*. The latter cuts down the number of outgoing edges from a given state of the model state space by performing a partition of the set of processes. This partition is based on their local state, and then only the edges corresponding to a single, representative process (the *leader* process) are constructed and stored for each equivalence class.

Similarly, this technique reduces the number of transitions departing from a given ES-LTS state to be explicitly considered due to the fact that, under suitable process equivalence conditions, all processes in the same equivalence class lead to the same traces in the ES-LTS; so, in the construction of the ES-LTS itself it is enough

to take only one process per class into account. Unlike [7, 21], this approach also deals with the symbolic data representation in the ES-LTS and is, therefore, suitable for the more powerful testing equivalence verification.

Of course, the notion of *process equivalence* must be made clear, bearing in mind that each process also has a context, consisting of the intruder knowledge K, the function Υ and the set of forbidden substitutions Λ; it follows that process equivalence will also involve context equivalence.

The notion of equivalence among processes proposed here is based on the syntactical identity of processes up to a substitution of both the constant and free variable names, that is to say, two processes P_i and P_j are equivalent iff there exists a suitable substitution which maps the names of constants and free variables of the first process on the constants and free variables of the second process.

Such a substitution induces a process equivalence only if the context is unaffected by the substitution: this trivially holds for the free variable names, in fact given a finite spi calculus process $(\nu\tilde{m})(P_1 \mid \ldots \mid P_n)$, where $(\nu\tilde{m})$ will be used as a shorthand notation for $(\nu\ m_1)\cdots(\nu\ m_n)$, the spi calculus semantics guarantees that $fv(P_i) \cap fv(P_j) = \emptyset$ for each $i \neq j$ with $i, j \in [1, \ldots, n]$.

More attention must be paid to constants, because it must be ensured that a substitution involving constants does not change the semantics of the behavior expression in force at the starting ES-LTS state. If this condition does not hold, there is no process equivalence, because the substitution could be indirectly *observed* by other processes.

The relation $\mathcal{R}$ informally described above can now be considered formally. It can be proved with ordinary effort that $\mathcal{R}$ is symmetric, reflexive and transitive: so, $\mathcal{R}$ is in fact an equivalence relation, and can be used to partition a parallel composition of spi calculus processes. Since spi calculus processes are finite, the partition of their parallel sub-processes can be computed algorithmically. Given a spi process $(\nu\tilde{m})(P_i \mid P_j \mid P)$ with context K, Λ and Υ, $\mathcal{R}$ is defined as

$$P_i\ \mathcal{R}\ P_j \quad \Longleftrightarrow \quad \exists\ \lambda_{ij}\ \mid \begin{cases} \lambda_{ij}\ \textit{is a bijective substitution} \\ P_i[\lambda_{ij}] = P_j \\ \lambda_{ij} = \lambda_{ij}^{-1} \\ Dom(\lambda_{ij}^{v}) = Im(\lambda_{ij}^{v}) = fv(P_i) \cup fv(P_j) \\ Dom(\lambda_{ij}^{c}) = Im(\lambda_{ij}^{c}) = fc(P_i) \cup fc(P_j) \\ K[\lambda_{ij}] = K, \quad \Lambda[\lambda_{ij}] = \Lambda, \quad \Upsilon[\lambda_{ij}] = \Upsilon \\ (\nu\tilde{m})(P_i \mid P_i[\lambda_{ij}] \mid P) = (\nu\tilde{m})(P_i \mid P_i[\lambda_{ij}] \mid P)[\lambda_{ij}] \end{cases} \tag{18}$$

Above, a substitution λ has been explicitly partitioned into two disjoint parts: λ^c involving only constant names, and λ^v involving only free variable names, when useful. Relations $Dom(\lambda_{ij}^{v}) = Im(\lambda_{ij}^{v}) = fv(P_i) \cup fv(P_j)$ and $Dom(\lambda_{ij}^{c}) = Im(\lambda_{ij}^{c}) = fc(P_i) \cup fc(P_j)$ mean that constant names replace constant names and variables replace variables.

The guarantee that the substitution does not affect the context is given by: $K[\lambda_{ij}] = K$, $\Lambda[\lambda_{ij}] = \Lambda$, and $\Upsilon[\lambda_{ij}] = \Upsilon$, which refers to the sets giving the state interpretation, and $(\nu\tilde{m})(P_i \mid P_i[\lambda_{ij}] \mid P) = (\nu\tilde{m})(P_i \mid P_i[\lambda_{ij}] \mid P)[\lambda_{ij}]$, which

states that the substitution leaves unchanged the syntactical form of the behavior expression of the state.

As for the actual implementation of the algorithm, the S^{3}A tool, for each pair of disjoint sub-processes P_i and P_j of the process associated with the current ES-LTS state, first checks their syntactical identity disregarding variable and constant names. If P_i and P_j pass this test, then the tool tries to construct the substitutions λ^v_{ij} and λ^c_{ij} by matching variable and constant names located in syntactically equivalent positions. As the last step, the tool checks whether the above-mentioned substitutions, λ^c_{ij} in particular, do not affect the context. When it finds multiple symmetries for the same ES-LTS state, the tool applies all of them as long as the sub-processes they operate upon are disjoint. When two or more symmetries apply to overlapping sub-processes, the tool selects the symmetry that acts on the "longest" sub-processes, where the length of a sub-process is the number of behavioral components it has.

For example, consider the following process:

$$P_1 \mid P_2 \triangleq c(x).c(y).c(z).[x\ is\ z]\ \overline{c}\langle y\rangle.0 \mid c(t).c(w).c(u).[t\ is\ u]\ \overline{c}\langle w\rangle.0\ ,$$

(with $K = \emptyset$, $\Upsilon = \emptyset$ and $\Lambda = \emptyset$). The substitution $\lambda = \langle t/x,\ w/y,\ u/z,\ x/t,\ y/w,\ z/u\rangle$ satisfies (18) and $P_1[\lambda] = P_2$ (and $P_1 = P_2[\lambda]$). If instead

$$P_1 \mid P_2 \triangleq c(x).c(y).c(z).[x\ is\ z]\ \overline{c}\langle y\rangle.0 \mid c(t).c(w).c(u).[w\ is\ u]\ \overline{c}\langle w\rangle.0$$

is considered, there is no substitution satisfying (18): in fact it is necessary to map w on y in the second input event, and on x in the test construct, so it is impossible to build an injective substitution. This occurs because in P_2 the comparison is between the second and the third input data, whereas P_1 compares the first and the third input data. Going on to consider a process:

$$(\nu\tilde{m})(P_1 \mid P_2 \mid P_3) \triangleq (\nu M)(\nu N)(c(x).\overline{c}\langle M\rangle.0 \mid c(y).\overline{c}\langle N\rangle.0 \mid \overline{c}\langle M\rangle.0))\ ,$$

with $K = \emptyset$, $\Upsilon = \emptyset$ and $\Lambda = \emptyset$: here $\lambda = \langle x/y,\ y/x,\ M/N,\ N/M\rangle$ gives $P_1 = P_2[\lambda]$, but $P_3[\lambda] \neq P_3$, thus the last requirement which was posed on the substitution,

$$(\nu\tilde{m})(P_i \mid P_i[\lambda_{ij}] \mid P) = (\nu\tilde{m})(P_i \mid P_i[\lambda_{ij}] \mid P)[\lambda_{ij}]\ ,$$

is violated. As a last example consider the process:

$$(\nu\tilde{m})(P_1 \mid P_2) \triangleq (\nu M)(\nu N)(c(x).\overline{c}\langle M\rangle.0 \mid c(y).\overline{c}\langle N\rangle.0)\ ,$$

with context $K = \{\langle c, l_0\rangle, \langle M, l_1\rangle\}$, $\Upsilon = \emptyset$ and $\Lambda = \emptyset$. A substitution $\lambda = \langle x/y,\ y/x,\ M/N,\ N/M\rangle$ gives the equivalence between P_1 and P_2, but the environment can distinguish M from N, since M is already in the attacker's knowledge. In fact, the requirement $K[\lambda] = K$ is violated in this case.

Theorem 1 *Given a spi process P with context K, Υ, Λ and a bijective substitution λ on constant and free variable names such that $\lambda = \lambda^{-1}$, then:*

$$(K \rhd P)_{\Upsilon,\Lambda} \xmapsto[\phi]{\mu} (K' \rhd P')_{\Upsilon',\Lambda'} \iff (K \rhd P)_{\Upsilon,\Lambda}[\lambda] \xmapsto[\phi]{\mu} (K' \rhd P')_{\Upsilon',\Lambda'}[\lambda]$$

The claim for reduction rules (5-6) and (9-17) must be proved separately. Considering for example the first rule (5), it has to be proved that:

$$\frac{K\vdash\sigma \qquad \langle\xi,\delta_\Lambda\rangle\in\Theta(\rho,K_{\Upsilon,\Lambda}) \qquad K'_{\Upsilon',\Lambda'}=f_{\langle\xi,\delta_\Lambda\rangle}(\rho,K_{\Upsilon,\Lambda})}{(K\rhd\overline{\sigma}\langle\rho\rangle.P)_{\Upsilon,\Lambda} \xrightarrow[\langle\xi,\delta_\Lambda\rangle[K'],\left\langle\delta^-_K(\rho),\ \delta^=_K(\rho),\ \rho\right\rangle[K']]{\overline{\sigma[\xi][K']}} (K'\rhd P[\xi])_{\Upsilon',\Lambda'}} \tag{5}$$

$$\Updownarrow$$

$$\frac{\begin{array}{c}K[\lambda]\vdash\sigma[\lambda] \quad \langle\xi[\lambda],\delta_\Lambda[\lambda]\rangle\in\Theta(\rho[\lambda],K[\lambda]_{\Upsilon[\lambda],\Lambda[\lambda]})\\ K'[\lambda]_{\Upsilon'[\lambda],\Lambda'[\lambda]}=f_{\langle\xi[\lambda],\delta_\Lambda[\lambda]\rangle}(\rho[\lambda],K[\lambda]_{\Upsilon[\lambda],\Lambda[\lambda]})\end{array}}{(K\rhd\overline{\sigma}\langle\rho\rangle.P)_{\Upsilon,\Lambda}[\lambda] \xrightarrow[\langle\xi,\delta_\Lambda\rangle[K'],\left\langle\delta^-_K(\rho),\ \delta^=_K(\rho),\ \rho\right\rangle[K']]{\overline{\sigma[\xi][K']}} (K'\rhd P[\xi])_{\Upsilon',\Lambda'}[\lambda]} \tag{19}$$

($\Longrightarrow$) By applying λ to $\overline{\sigma}\langle\rho\rangle.P$ and its context in (5):

$$\frac{\begin{array}{c}K[\lambda]\vdash\sigma[\lambda] \quad \langle\xi_\lambda,\delta_{\Lambda_\lambda}\rangle\in\Theta(\rho[\lambda],K_{\Upsilon[\lambda],\Lambda[\lambda]})\\ K'_{\lambda}{}_{\Upsilon'_\lambda,\Lambda'_\lambda}=f_{\langle\xi_\lambda,\delta_{\Lambda_\lambda}\rangle}(\rho[\lambda],K[\lambda]_{\Upsilon[\lambda],\Lambda[\lambda]})\end{array}}{(K\rhd\overline{\sigma}\langle\rho\rangle.P)_{\Upsilon,\Lambda}[\lambda] \xrightarrow[\left\langle\delta^-_{K[\lambda]}(\rho[\lambda]),\ \delta^=_{K[\lambda]}(\rho[\lambda]),\ \rho[\lambda]\right\rangle[K'_\lambda]]{\overline{\sigma[\lambda][\xi_\lambda][K'_\lambda]}} (K'_\lambda\rhd P[\lambda][\xi_\lambda])_{\Upsilon'_\lambda,\Lambda'_\lambda}} \tag{20}$$

In order to prove this implication it is necessary to obtain (19) from (20). In particular, being λ bijective and $\lambda=\lambda^{-1}$, it is not difficult to prove that:

$$\begin{array}{rcl}K\vdash\sigma & \Leftrightarrow & K[\lambda]\vdash\sigma[\lambda]\\ \langle\xi,\delta_\Lambda\rangle\in\Theta(\rho,K_{\Upsilon,\Lambda}) & \Leftrightarrow & \langle\xi[\lambda],\delta_{\Lambda[\lambda]}\rangle\in\Theta(\rho[\lambda],K_{\Upsilon[\lambda],\Lambda[\lambda]})\\ K'_{\Upsilon',\Lambda'}=f_{\langle\xi,\delta_\Lambda\rangle}(\rho,K_{\Upsilon,\Lambda}) & \Leftrightarrow & K'[\lambda]_{\Upsilon'[\lambda],\Lambda'[\lambda]}=f_{\langle\xi[\lambda],\delta_{\Lambda[\lambda]}\rangle}(\rho[\lambda],K[\lambda]_{\Upsilon[\lambda],\Lambda[\lambda]})\\ P[\lambda][\xi[\lambda]] & = & (P[\xi])[\lambda]\end{array}$$

so that $\xi_\lambda=\xi[\lambda]$, $\delta_{\Lambda_\lambda}=\delta_{\Lambda[\lambda]}$, $K'_\lambda=K'[\lambda]$, $\Upsilon'_\lambda=\Upsilon'[\lambda]$ and $\Lambda'_\lambda=\Lambda'[\lambda]$. Thus (20) becomes:

$$\frac{\begin{array}{c}K[\lambda]\vdash\sigma[\lambda] \quad \langle\xi[\lambda],\delta_{\Lambda[\lambda]}\rangle\in\Theta(\rho[\lambda],K_{\Upsilon[\lambda],\Lambda[\lambda]})\\ K'[\lambda]_{\Upsilon'[\lambda],\Lambda'[\lambda]}=f_{\langle\xi[\lambda],\delta_{\Lambda[\lambda]}\rangle}(\rho[\lambda],K[\lambda]_{\Upsilon[\lambda],\Lambda[\lambda]})\end{array}}{(K\rhd\overline{\sigma}\langle\rho\rangle.P)_{\Upsilon,\Lambda}[\lambda] \xrightarrow[\left\langle\delta^-_{K[\lambda]}(\rho[\lambda]),\ \delta^=_{K[\lambda]}(\rho[\lambda]),\ \rho[\lambda]\right\rangle[K'[\lambda]]]{\overline{\sigma[\lambda][\xi[\lambda]][K'[\lambda]]}} (K'\rhd P[\xi])_{\Upsilon',\Lambda'}[\lambda]} \tag{21}$$

Now it has to be proved that the transition label of (21) is equal to the label of (19), i.e.:

$$\xrightarrow[\left\langle\delta^-_{K[\lambda]}(\rho[\lambda]),\ \delta^=_{K[\lambda]}(\rho[\lambda]),\ \rho[\lambda]\right\rangle[K'[\lambda]]]{\overline{\sigma[\lambda][\xi[\lambda]][K'[\lambda]]}} = \xrightarrow[\langle\xi,\delta_\Lambda\rangle[K'],\left\langle\delta^-_K(\rho),\ \delta^=_K(\rho),\ \rho\right\rangle[K']]{\overline{\sigma[\xi][K']}}$$

but this is easily done since $\theta[\lambda][K[\lambda]]=\theta[K]$ holds for any spi term θ.
($\Longleftarrow$) Relies on the fact that $\lambda=\lambda^{-1}$.

The proof for the remaining reduction rules can be carried out in a similar way.

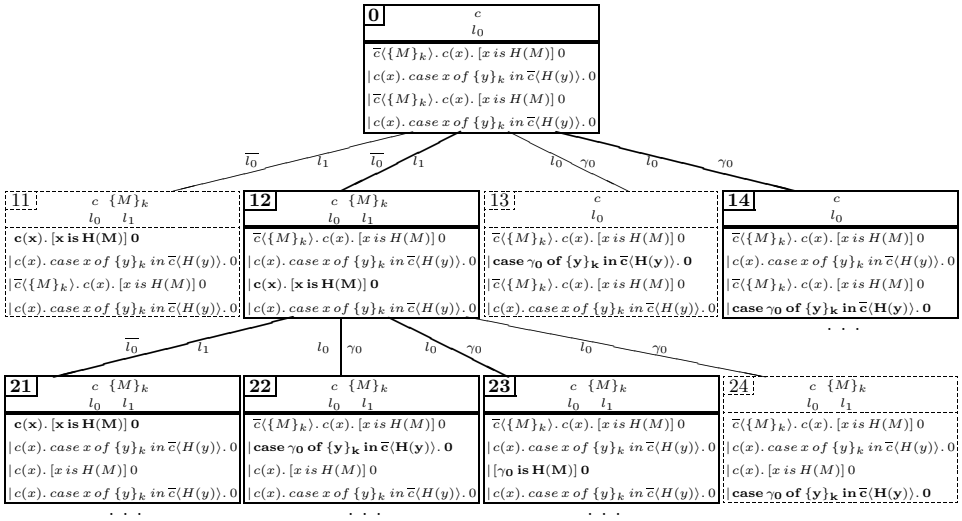

Figure 2: A quite simple protocol: partial ES-LTS with states and traces.

Theorem 2 *Given a context K, Υ, Λ, a process $P \triangleq (\nu\tilde{m})(P_i \mid P_i[\lambda_{ij}] \mid P)$, and a substitution λ_{ij} defined as in (18), then:*

$$\begin{array}{ccc} (K \rhd (\nu\tilde{m})(P_i \mid P_i[\lambda_{ij}] \mid P))_{\Upsilon,\Lambda} & \xrightarrow[\phi]{\mu} & (K' \rhd (\nu\tilde{m})(P_i' \mid P_i[\lambda_{ij}] \mid P'))_{\Upsilon',\Lambda'} \\ \Updownarrow & & \\ (K \rhd (\nu\tilde{m})(P_i \mid P_i[\lambda_{ij}] \mid P))_{\Upsilon,\Lambda} & \xrightarrow[\phi]{\mu} & (K' \rhd (\nu\tilde{m})(P_i' \mid P_i[\lambda_{ij}] \mid P'))_{\Upsilon',\Lambda'}[\lambda_{ij}] \end{array}$$

Because of (18): $K[\lambda_{ij}] = K$, $\Lambda[\lambda_{ij}] = \Lambda$, $\Upsilon[\lambda_{ij}] = \Upsilon$, $(\nu\tilde{m})(P_i \mid P_i[\lambda_{ij}] \mid P) = (\nu\tilde{m})(P_i \mid P_i[\lambda_{ij}] \mid P)[\lambda_{ij}]$, and $\lambda_{ij} = \lambda_{ij}^{-1}$, so it follows that:

$$(K \rhd (\nu\tilde{m})(P_i \mid P_i[\lambda_{ij}] \mid P))_{\Upsilon,\Lambda} = (K \rhd (\nu\tilde{m})(P_i \mid P_i[\lambda_{ij}] \mid P))_{\Upsilon,\Lambda}[\lambda_{ij}]\ ,$$

and the hypotheses of Theorem 1 hold. Theorem 2 states that, under suitable hypotheses, two identical transitions that originate from the same state and lead to two states that differ only by a substitution of constant and free variable names, i.e. where hypotheses of Theorem 1 still hold, are allowed. Thus, Theorem 2 can be recursively applied to obtain two (sub)-ES-LTSs with the same transition labels and corresponding states, barring a substitution. In practice, since one of the two (sub)-ES-LTSs has exactly the same labels as the other, the redundant (sub)-ES-LTS can be omitted, both in generating and comparing traces.

5. An Example

Fig. 2 shows how this reduction technique works on two parallel sessions of a simple protocol inspired by [10], where two agents A and B share a secret key k and exchange two messages. The initial intruder knowledge is assumed to be $K = \{\langle c, l_0\rangle\}$. In Fig. 2 each ES-LTS state is represented by a box showing the

intruder knowledge K (upper part) and the spi calculus specification of the process P (lower part). Thus, the spi calculus specification of the protocol as a whole can be found in the lower part of the box that corresponds to state 0. Set Λ is always empty in this case, so it is not shown, and Υ is not explicitly represented since it can be easily deduced from the state where each generic term was generated. Each arc connecting a pair of states has been labelled with μ and ϕ, on the left and right side respectively. In this example ξ and δ_Λ are always empty in the complementary action ϕ of output transitions, and δ_K is the same as the output term ρ, so ϕ is simply represented as $\rho[K']$. Moreover, internal events on channels known to the intruder are not represented, since they only generate pure τ labels and do not contribute to traces. Dashed states and their corresponding thin input lines are those identified as redundant by this technique. Dots under *thick* states mean that the evolution of the ES-LTS produces other states not shown in the figure.

State 0 can be partitioned into a single equivalence class, containing two instances of $\overline{c}\langle\{M\}_k\rangle.\ c(x).\,[x\,is\,H(M)]\,0 \quad | \quad c(x).\,case\,x\,of\,\{y\}_k\,in\,\overline{c}\langle H(y)\rangle.\,0$. The substitution simply maps each x and y of the first instance onto the corresponding ones of the second. It should be noted that the free variables x and y of one instance are different from their corresponding counterparts, because of their different scope.

The output event (action) performed by the first instance leads to the state marked 11 in Fig. 2, whereas state 12 is reached by considering the corresponding event of the second instance. The labels of the two transitions are the same (output of message l_1 on channel $\overline{l_0}$) and the two states differ only in the role of their free variables, i.e. they are the same state except for a substitution of the variable names. Thus, the sub-traces generated starting with state 11 are the same as those starting from state 12. The same reasoning applies to states 13 and 14, obtained by considering the input event (action) of the two instances. Here a generic term γ_0 is created by the input action on channel l_0.

Starting with state 12, there are three equivalence classes: two of them are made up by $\overline{c}\langle\{M\}_k\rangle.\,c(x).\ [x\,is\,H(M)]\,0$ and $c(x).\,[x\,is\,H(M)]\,0$ respectively, whereas the third one contains two instances of $c(x).\,case\,x\,of\,\{y\}_k\,in\,\overline{c}\langle H(y)\rangle.\,0$. These two instances lead to states 22 and 24, where a similar reasoning can be applied.

6. Experimental Results and Related Work

A preliminary version of a tool, called S^{3}A, that fully implements the technique described in [9] and the enhancement discussed in this paper, has been used to verify several cryptographic protocols and, in particular, to test the efficiency of the proposed symmetry-based reduction technique. Without symmetry-based reduction, the ES-LTS of Fig. 2 has 2,206,186 states, while their number drops to 215,268 when symmetry-based reductions are used (less than 10%).

Moreover, this approach has been compared with other tools implementing similar reduction techniques for cryptographic protocols analysis. To the best of our knowledge, only the BRUTUS model checker [8] implements symmetry-based reduction techniques to speed up the verification of security properties of cryptographic protocols; both S^{3}A and BRUTUS carry out the whole analysis, including symmetry-

Table 2: Experimental results.

			BRUTUS		S^3A		
Prot.	Init	Resp	None	Symm	None	Symm	Time
N-S	1	1	1,208	1,208	81	81	< 1s
N-S	1	2	1,227,415	613,713	36,233	17,365	2s
N-S	2	2	X	X	9,007,163	2,176,344	195s
N-S	2	3	X	X	X	390,126,070	94,955s
WMF	3	3	X	X	X	40,959,126	50s

based reductions, in a fully automated way. Moreover, BRUTUS also uses partial order reductions, to further cut down the number of states, but since interest here is in symmetries, this paper limits the analysis to this topic.

In [8] numerical results are given for the analysis of three popular protocols, namely 1KP [3], Needham-Schroeder with public key (N-S) [18] and the Wide Mouthed Frog protocol (WMF) [5]. For the purposes of this paper, the analysis of the last two protocols is more interesting, since it has been carried out with an increasing number of instances for each role, giving ideas of the exponential growth of the number of states when the number of instances of each role increases.

The first column of Table 2 shows the protocol name acronym and the number of instances of the initiator and responder roles respectively. The second column shows the number of states generated by BRUTUS when no reductions are applied (*None* sub-column) and when symmetry-based reductions are used (*Symm* sub-column).

The rightmost column lists the results obtained with S^3A. An '*X*' symbol is used when the number of states is too big to be computed with reasonable resources ($>$ 700,000,000 states). For S^3A, the *Time* sub-column also lists the execution times, in seconds, when the tool is run on an Athlon XP 2100+ PC.

The comparison is based on the number of states instead of execution time because, to the best of our knowledge, no execution time information is publicly available for BRUTUS.

Although BRUTUS also implements partial order reductions whose results are not depicted here, it must be pointed out that, when the comparison is carried out using the same reduction technique, S^3A behaves undoubtedly better.

The second and third row of Table 2 show that the *compression ratio* on the total number of states achieved by symmetry-based reductions is about $1 : 2$ when both tools work on the same problem "N-S 1 2" ($613,713/1,227,415$ and $17,365/36,233$), and the performance of S^3A improves, reaching $1 : 4.14$, when the problem size increases, as in "N-S 2 2" ($2,176,344/9,007,163$). This result demonstrates that symmetry-based reductions perform better when the number of instances of roles grows, and more symmetries can be exploited. For the same reason, it is easy to understand why they do not yield any advantage for "N-S 1 1".

It can also be noted that S^3A performs better than BRUTUS in absolute terms, both with and without the help of symmetry-based reductions: in fact the ratio between the number of states generated by the tools falls between $1 : 15$ ($81 : 1,208$)

and $1:35$ $(17,365:613,713)$.

The difference is mainly due to the fact that BRUTUS is a *concrete* model checker, i.e. when the process performs an input action, all messages that the intruder can build starting from its knowledge have to be explicitly considered. This number is infinite even if the knowledge is finite, thus the exhaustive message generation is restricted by means of an artificial upper limit on the size of the messages the intruder is enabled to generate. Although this limit makes the problem tractable, it potentially implies a restriction on the attacks that the technique can detect, and input event management still remains a critical point for the explosion of the number of states. On the other hand, S^3A adopts a symbolic representation of input messages, with a twofold advantage: input events are not a potential point of state explosion, and having no limitations on message lengths and so on, our state generation is exhaustive. Moreover, S^3A also deals with non-atomic keys and checks safety properties by means of testing equivalence verification, which allows secrecy properties to be formulated in a more accurate way than with systems like BRUTUS.

For what concerns overheads, the experiments carried out on an instrumented version of the tool show that S^3A spends less than 10% of its overall execution time dealing with symmetry-based reductions, a figure more than acceptable given the overall performance gains they give. To the best of our knowledge, no information about the overheads of symmetry and partial order reductions is publicly available for BRUTUS, although the algorithms used to carry them out are thoroughly described in [8].

7. Conclusions

The results presented in this paper extend those achieved previously in the field of automatic verification of security protocols, because they provide evidence that there is a viable alternative to the use of theorem proving for the verification of complex security properties based on testing equivalence.

With respect to [4], the step-by-step knowledge equivalence verification is no longer needed, because the ES-LTS transition labels introduced in [9] incorporate all the information needed to verify testing equivalence. Moreover, [4] deals with a spi calculus dialect where public-key encryption, hashing, integers, and non-atomic keys are not considered.

In addition, symmetries arising from multiple parallel sessions are exploited by a reduction technique which limits the size of the model to be checked and can be implemented in a fully automated way with acceptable overheads with respect to the analysis process as a whole.

The advantages of this technique, which are difficult to theoretically quantify, have been verified with the S^3A tool: its underlying theoretical framework [9] is more sophisticated and complex than many others, since it deals with testing equivalence, thus allowing a finer grain analysis of secrecy properties than is possible with reachability analysis. Moreover, it does not suffer from drawbacks such as handling only atomic keys and/or limitations on the size of messages and so on. Despite this

greater generality, S^3A has provided encouraging results, even better than those obtained using other tools based on a simpler and more limited theoretical approach.

Further improvements of the technique presented here can be achieved by defining other testing equivalence preserving reductions such as, for example, reductions based on partial order, or by extending the technique to deal with sub-expressions that are equal up to the substitution of generic terms.

Acknowledgements

This work was developed within the framework of the CNR project "Metodi e strumenti per la progettazione di sistemi software-intensive ad elevata complessità". The authors wish to thank the anonymous reviewers, whose valuable comments and suggestions helped to improve the quality of this paper.

References

1. M. Abadi and A. D. Gordon, "A bisimulation method for cryptographic protocols," *Nordic J. Comput.* **5** (1998) 267–303.
2. M. Abadi and A. D. Gordon, "A calculus for cryptographic protocols the spi calculus," SRC - Research Report 149, Digital System Research Center, 1998.
3. M. Bellare, J. Garay, R. Hauser, A. Herberg, H. Krawczyk, M. Steiner, G. Tsudik, and M. Waidner, "*i*KP - A family of secure electronic payment protocols," *Proc. 1st USENIX Workshop on Electronic Commerce*, Berkeley, CA, USENIX Assoc., 1995, pp. 157–166.
4. M. Boreale, R. De Nicola, and R. Pugliese, "Proof techniques for cryptographic processes," *SIAM J. Comput.* **31** (2002) 947–986.
5. M. Burrows, M. Abadi, and R. Needham, "A logic of authentication," *Proceedings of the Royal Society, Series A* **426** (1989) 233–271.
6. I. Cibrario Bertolotti, L. Durante, R. Sisto, and A. Valenzano, "A new knowledge representation strategy for cryptographic protocol analysis," *Proc. Tools and Algoritms for the Construction and Analysis of Systems (TACAS 2003)*, Berlin, Lecture Notes in Computer Science, vol. 2619, Springer-Verlag, 2003, pp. 284–298.
7. E. M. Clarke, S. Jha, and W. Marrero, "Partial order reductions for security protocol verification," *Proc. Tools and Algoritms for the construction and Analysis of Systems (TACAS 2000)*, Berlin, Lecture Notes in Computer Science, vol. 1785, Springer-Verlag, 2000.
8. E. M. Clarke, S. Jha, and W. Marrero, "Verifying security protocols with Brutus," *ACM Trans. Softw. Eng. Meth.* **9** (2000) 443–487.
9. L. Durante, R. Sisto, and A. Valenzano, "Automatic testing equivalence verification of spi calculus specifications," *ACM Trans. Softw. Eng. Meth.* **12** (2003) 222–284.
10. M. Fiore and M. Abadi, "Computing symbolic models for verifying cryptographic protocols," *Proc. 14th IEEE Computer Security Foundations Workshop (CSFW 2001)*, Washington, IEEE Computer Society Press, 2001, pp. 160–173.
11. S. Gnesi, D. Latella, and G. Lenzini, "A BRUTUS logic for the Spi-Calculus," *Proc. WITS'02*, 2002.
12. K. Jensen, *Coloured Petri nets: basic concepts, analysis methods and practical use* (Springer-Verlag, Berlin, 1995).
13. G. Lowe, "Breaking and fixing the Needham-Schroeder public-key protocol using

FDR," *Proc. Tools and Algoritms for the Construction and Analysis of Systems (TACAS 1996)*, Berlin, Lecture Notes in Computer Science, vol. 1055, Springer-Verlag, 1996, pp. 147–166.

14. G. Lowe, "Some new attacks upon security protocols," *Proc. 9th IEEE Computer Security Foundations Workshop (CSFW 1996)*, Washington, IEEE Computer Society Press, 1996, pp. 162–169.
15. G. Lowe, "Casper: a compiler for the analysis of security protocols," *Proc. 10th IEEE Computer Security Foundations Workshop (CSFW 1997)*, Washington, IEEE Computer Society Press, 1997, pp. 18–30.
16. J. K. Millen, S. C. Clark, and S. B. Freedman, "The Interrogator: Protocol security analysis," *IEEE Trans. Softw. Eng.* **13** (1987) 274–288.
17. R. Milner, J. Parrow, and D. Walker, "A calculus of mobile processes, parts I and II," *Inf. Comput.* **100** (1992) 1–77.
18. R. Needham and M. Schroeder, "Using encryption for authentication in large networks of computers," *Communications of the ACM* **21** (1978) 993–999.
19. L. C. Paulson, "The inductive approach to verifying cryptographic protocols," *J. Comput. Sec.* **6** (1998) 85–128.
20. S. Schneider, "Verifying authentication protocols in CSP," *IEEE Trans. Softw. Eng.* **24** (1998) 741–758.
21. A. P. Sistla, V. Gyuris, and E. A. Emerson, "Smc: A symmetry-based model checker for verification of safety and liveness properties," *ACM Trans. Softw. Eng. Meth.* **9** (2000) 133–166.