Modeling and analysis of security protocols using role based specifications and Petri nets

Roland Bouroulet, Raymond Devillers, Hanna Klaudel, Elisabeth Pelz, Franck Pommereau

Modeling and analysis of security protocols using role based specifications and Petri nets

Roland Bouroulet*, Raymond Devillers†,
Hanna Klaudel‡, Elisabeth Pelz* and Franck Pommereau*

*) LACL, Université Paris Est, 61 av. du Gén. de Gaulle, F-94010 Créteil.
†) Département d'Informatique, CP212, Université Libre de Bruxelles.
‡) IBISC, Université d'Evry, bd François Mitterrand, F-91025 Evry.

Abstract. In this paper, we introduce a framework composed of a syntax and its compositional Petri net semantics, for the specification and verification of properties (like authentication) of security protocols. The protocol agents (*e.g.*, an initiator, a responder, a server, a trusted third party, ...) are formalized as *roles*, each of them having a predefined behavior depending on their global and also local knowledge (including for instance public, private and shared keys), and may interact in a potentially hostile *environment*.
The main characteristics of our framework, is that it makes explicit, structured and formal, the usually implicit information necessary to analyse the protocol, for instance the public and private *context* of execution. The roles and the environment are expressed using SPL processes and compositionally translated into high-level Petri nets, while the context specifying the global and local knowledge of the participants in the protocol is used to generate the corresponding initial marking (with respect to the studied property). Finally, this representation is used to analyse the protocol properties, applying techniques of simulation and model-checking on Petri nets. The complete approach is illustrated on the case study of the Kao-Chow authentication protocol.

Key words: security protocols, formal specification, process algebras, Petri nets.

1 Introduction

In the last years, security protocols have become more and more studied for their behaviours, causal dependencies and secrecy properties. Such protocols, where messages are asynchronously exchanged via a network, assume an underlying infrastructure, composed of (asymmetric) public, private and (symmetric) shared keys [30, 31]. In order to specify more precisely security protocols, formal description languages have been proposed, like the *Spi-Calculus* in [3] or the *Security Protocol Language* (*SPL*) in [14]. The latter is an economical process language inspired from process algebras like the asynchronous π-Calculus [26]: each process is defined by a term of a specialized algebra which allows to represent sequences and parallel compositions of (more or less complex) input and

output actions which may contain messages. Thus, in contrast to other notations, sending and receiving messages can and have to be explicitly specified. However, when used to specify protocols with various agents, each of them playing a proper role with respect to the (global, but also local) knowledge it may have, an SPL presentation needs to be complemented in order to be verified. Usually, a number of implicit information is added, concerning for instance the global context of the protocol execution, the number and the identity of the participants, the knowledge about encryption keys each of them has, and so on.

In this paper, we provide a solution allowing to take into account all this important information in a structured way, making it explicit. We focus on the specification and verification of properties (like authentication) of security protocols. The active elements of these protocols may be formalized as *roles*, like for instance an initiator, a responder or a server, each of them having a predefined behavior taking place in a given *context*. They should interact in some potentially hostile *environment* in such a way that desired properties are preserved. Thus, in our approach, we define a framework in which a protocol is specified as a triple composed of a number of roles, an environment and a context. The roles and the environment are expressed using SPL processes, while the context specifies the knowledge of the participants in the protocol (including public, private and shared keys).

Next, this specification is translated into a class of composable high-level Petri nets. In particular, the roles and the environment are translated into *S-nets* [9, 10]. All these nets are composed in parallel in order to get a single net. The context serves to define its initial marking (with respect to the studied property). Finally, this representation is used to analyse the protocol properties, applying techniques of simulation and model-checking [19].

With respect to previous translations into Petri nets [9, 10], the present paper also introduces a way to use secret shared keys cryptography (while only public key cryptography was available before), and to take into account information about the past (like compromised old-session keys).

1.1 Case study

The approach will be illustrated on the Kao-Chow (KC) authentication protocol [21, 12], chosen as running example, involving participants exchanging messages using shared encryption keys. It implements three roles: those for an initiator, a responder and a server. As usual in a cryptographic context, all the participants make use of nonces, which are newly created values used by the participants to sign their messages.

The KC protocol between an initiator A, a responder B and a server S, allows A and B to mutually authenticate and to share a symmetric key K_{ab} used for secure communications after the agreement, *i.e.*, at the end of the protocol A and B are sure about the secrecy of their knowledge about K_{ab}. Informally and assuming that K_{as} and K_{bs} are symmetric keys initially known only by A and S, and respectively by B and S, the KC protocol may be described as follows:

1. $A \rightarrow S : A, B, m$
   The initiator agent A sends to the server S its own agent name A, the name of the agent B with which it would like to communicate, together with a nonce m.
2. $S \rightarrow B : \{A, B, K_{ab}, m\}_{K_{as}}, \{A, B, K_{ab}, m\}_{K_{bs}}$
   Then, S sends to B a message containing the received names A and B and the nonce m, together with the generated symmetric key K_{ab}, first encrypted with K_{as}, the symmetric key it shares with A, second with K_{bs} the one it shares with B. This second part, B should decrypt after reception.
3. $B \rightarrow A : \{A, B, K_{ab}, m\}_{K_{as}}, \{m\}_{K_{ab}}, n$
   Then, B sends to A the first part of the received message (that it cannot decrypt because it is encrypted with K_{as}), together with the received nonce m encrypted with the shared (newly obtained) key K_{ab}, and its own nonce n. After reception, A decrypts the first part of the message, gets the key K_{ab} and checks it by decrypting its own nonce m.
4. $A \rightarrow B : \{n\}_{K_{ab}}$
   Finally, A sends back to B the nonce n encrypted with K_{ab}.

This protocol suffers a similar kind of attack as the Denning Sacco freshness attack on the Needham Schroeder symmetric key protocol, when an older session symmetric key K_{ab} has been compromised.

1. $A \rightarrow S : A, B, m$
2. A and B previously used the symmetric key K_{ab} and we assume from now on that K_{ab} is compromised. So the following exchange has already occurred and can be replayed: $S \rightarrow B : \{A, B, m, K_{ab}\}_{K_{as}}, \{A, B, m, K_{ab}\}_{K_{bs}}$. An attacker I may now impersonate S and A to force B to reuse K_{ab} in a communication with what it believes to be A, while it is I.
3. $I(S) \rightarrow B : \{A, B, m, K_{ab}\}_{K_{as}}, \{A, B, m, K_{ab}\}_{K_{bs}}$
   Agent I behaves as S and replays the same message to B.
4. $B \rightarrow I(A) : \{A, B, m, K_{ab}\}_{K_{as}}, \{m\}_{K_{ab}}, n'$
   B executes the next step of the protocol and sends a message with a nonce n' to I, believing it is A.
5. $I(A) \rightarrow B : \{n'\}_{K_{ab}}$
   I, knowing the symmetric key K_{ab}, encrypts n' and sends it back to B. Now B believes to have authenticated A and pursues a secret communication with I instead of A.

The corresponding role based specification of this protocol comprises three roles (for one initiator, one server and one responder), each of them being formalized as an SPL process together with the knowledge about keys that it knows. The environment is represented by a generic SPL process simulating a potentially agressive attacker, and the global context is empty as the protocol does not assume any public knowledge. This specification is the basis of the compositional translation to high-level Petri nets for model-checking and simulation.

1.2 Outline

The paper is structured as follows: The next section presents syntactical aspects of our framework of role based specification, and their intuitive meaning. It includes a short presentation of the SPL syntax and its intuitive semantics, augmented by the explicit expression of keys, and the formal definition of a role based specification which uses SPL. The three elements of such a specification are described next, namely the initialized roles, the environment and the public context. This syntactical part is illustrated by providing a role based specification of the KC protocol, and its variants depending on the global context definition.

The next section introduces our Petri net (S-net) semantics of a role based specification and its compositional construction. In particular, the translation algorithm from SPL terms to S-nets for roles and for the environment is detailed as well as the one giving to the nets their initial marking. Dealing with symmetric keys, and possibly several of them between one pair of agents, needs a careful semantical treatment, which is explained and illustrated on small examples. Also, because of the modeling of a fully general environment, the translation potentially leads to infinite nets. Various ways allowing to cope with this problem are then presented and discussed.

Then, a section illustrates the use of the role based specification in order to automatically obtain a Petri net representation of KC protocol, and relates our verification experiences using existing model-checking tools.

Finally, we conclude by giving some indications about the directions of our future work.

2 Role based specification with SPL

First, we briefly recall the syntax and the intuitive semantics of SPL agents, inspired from [14].

2.1 Syntax and intuitive semantics of SPL

We assume countably infinite disjoint sets:

- of nonces (which are assumed to be numerical values) $N = \{n, n', n'', \ldots\}$;
- of agents $G = \{A, B, \ldots\}$;
- and of indexes for processes and keys $I = \{i, \ldots\}$, containing in particular the natural numbers $\mathbb{N}$.

We distinguish also three disjoint sets of variables:

- $V_N = \{u, v, w, x, y, z, \ldots\}$ for nonce variables;
- $V_G = \{X, Y, Z, \ldots\}$ for agent variables;
- and $V_M = \{\Psi, \Psi', \Psi_1, \ldots\}$ for message variables.

We use the vector notation $\overrightarrow{a}$ which abbreviates some possibly empty list of variables $a_1, \ldots, a_l$ (in $V_N \cup V_G \cup V_M$), usually written in SPL as $\{a_1, \ldots, a_l\}$. Also, the notation $[\overrightarrow{V}/\overrightarrow{a}]$ represents the componentwise sort-preserving substitutions $a_i \mapsto V_i$, for $i \in \{1, \ldots, l\}$.

The syntax of SPL is given by the following grammar:

$$
\begin{array}{rll}
g ::= & A, ... \mid X... & \text{agent expressions} \\
e ::= & n, ... \mid u... & \text{nonce expressions} \\
k ::= & \langle Pub, e, g\rangle \mid \langle Priv, e, g\rangle \mid \langle Sym, e, g_1, g_2\rangle & \text{key expressions} \\
M ::= & g \mid e \mid k \mid M_1, M_2 \mid \{M\}_k \mid \Psi & \text{messages} \\
p ::= & \mathit{out}\ \mathit{new}\ \overrightarrow{u}\ M.p \ \mid\ \mathit{in}\ \mathit{pat}\,\overrightarrow{a}\ M.p \mid \|_{i\in I}\ p_i & \text{processes}
\end{array}
$$

where

(a) $\langle Pub, e, g\rangle$, $\langle Priv, e, g\rangle$, $\langle Sym, e, g_1, g_2\rangle$, where e is the actual value of the key, are used respectively for the public key of g, the private key of g, and the symmetric key shared by g_1 and g_2; we may also use indexes like $\langle Pub, e, g, i\rangle$, $\langle Priv, e, g, i\rangle$, $\langle Sym, e, g_1, g_2, i\rangle$, $i \in I$, if we need to manage several similar keys simultaneously. We shall denote by K the set of all possible private, public and symmetric keys without variables in the SPL format.

(b) A message may be a name, a nonce, a key, a composition of two messages (M_1, M_2), an encrypted message M using a key k, which is written $\{M\}_k$, or a message variable. By convention, messages are always enclosed in brackets $\{\ \}$, thus allowing to distinguish between a nonce n and a message $\{n\}$ only containing it. Moreover, we denote by Msg the set of all messages without variables formed with respect to the SPL grammar, and $\mu(\mathsf{Msg})$ the set of all multi-sets over it.

(c) In the process $\mathit{out}\ \mathit{new}\ \overrightarrow{u}\ M.p$, the out-action prefixing p chooses fresh distinct nonces $\overrightarrow{n} = \{n_1, \cdots, n_l\}$, binds them to variables $\overrightarrow{u} = \{u_1, \ldots, u_l\}$ and sends the message $M[\overrightarrow{n}/\overrightarrow{u}]$ out to the network, then the process resumes as $p[\overrightarrow{n}/\overrightarrow{u}]$. It is assumed that all the variables occurring in M also occur in $\overrightarrow{u}$. The *new* construct, like in [14], ensures the freshness of values in $\overrightarrow{n}$. Notice that communications are considered as *asynchronous*, which means that output actions do not need to wait for inputs. By convention, we may simply omit the keyword *new* in an out-action if the list $\overrightarrow{u}$ of 'new' variables is empty.

(d) In the process $\mathit{in}\ \mathit{pat}\ \overrightarrow{a}M.p$, the in-action prefixing p awaits for an input that matches the pattern M for some (sort-preserving) binding of the pattern variables $\overrightarrow{a}$, then the process resumes as p under this binding, *i.e.*, as $p[\overrightarrow{\alpha}/\overrightarrow{a}]$, where $\overrightarrow{\alpha}$ are the values in $N \cup G \cup \mathsf{Msg}$ received in the input message. It is required that $\overrightarrow{a}$ is an ordering of all the pattern variables occurring in M. We may also omit the keyword *pat* in an in-action if the list $\overrightarrow{a}$ of 'new' variables is empty.

(e) The process $\|_{i\in J}\ p_i$ is the parallel composition of all processes p_i with $i \in J \subseteq I$. The particular case where J is empty defines the process $nil = \|_{i\in\emptyset}\ p_i$;

all processes should end with *nil* but, by convention, we usually omit it in examples. Replication of a process p, denoted $!p$, stands for an infinite composition $\|_{i\in\mathbb{N}}\ p$.

We shall denote by $var(p)$ the set of all the free variables occurring in the process p, *i.e.*, the variables used in p which are not bound by an in or out prefix. In order to simplify the presentation, we assume that all the sets $\overrightarrow{u}$ and $\overrightarrow{a}$ of variables occurring in the out and in prefixes of processes, as well as their sets of free variables, are pairwise disjoint.

2.2 Role based specification

We can now introduce our approach which mostly relies on notions like "role", "environment", and private and public "contexts".

Given an SPL grammar as above, a *role* (*i.e.*, a named process) is a definition of the form $\delta(a_1, \ldots, a_{m_\delta}) \stackrel{\text{df}}{=} p_\delta$, where p_δ is an SPL process, such that $var(p_\delta) \subseteq \{a_1, \ldots, a_{m_\delta}\}$. For short, such a role is often identified with its name δ. Its context $context_\delta$ is defined as a pair (f_δ, l_δ), where

- f_δ: $\{a_1, \ldots, a_{m_\delta}\} \to G \cup N \cup \mathsf{Msg}$ is a sort-preserving mapping that associates an agent to each agent variable, a nonce (value) to each nonce variable and a message to each message variable;
- and $l_\delta \subseteq K$ is a subset of keys known by the role δ.

Intuitively, δ is the name of an agent, p_δ describes its behaviour, f_δ defines the instantiation of the process by fixing the values of the free variables and l_δ specifies the initial private knowledge of the agent (usually the agents initially know some keys; during the execution of the protocol they will be able to learn some more keys from the messages received through the network). We shall denote by $(\delta(f_\delta), l_\delta)$ an initialised role δ together with its private context.

Besides agents, the system will also encompass some spies and the network interconnecting all of them.

A *role based specification* is defined as a triple

$$(\mathit{Initialized\_Roles},\quad \mathit{Environment},\quad \mathit{Public\_Context}),$$

where

- $\mathit{Initialized\_Roles} \subset \{(\delta(f_\delta), l_\delta) \mid \delta \text{ is a role}\}$ is a (finite) non-empty set of initialised roles with their private contexts;
- $\mathit{Environment} \subset \{s \mid s \text{ is a role}\}$ defines an environment, *i.e.*, a set of prototype roles (spies) whose replication and composition will allow to form more complex attackers; note that spies do not have a context, since they have no free variable, nor private knowledge (in order to cooperate, they make public all they know);
- $\mathit{Public\_Context} \in \mu(\mathsf{Msg})$ is the initial public context, *i.e.*, a (multi-)set of messages previously sent on the network and available to everyone (to spies as well as to agents) when the protocol starts; it comprises usually the names of the agents in presence, but also possibly compromised keys, etc.

Using a role based specification to express a security protocol hence consists in defining the roles corresponding to all the participants (initiator, responder, trusted third party, ...) and the roles corresponding to the prototypes of the potential attackers (environment). Potential attackers may be described using *spy* processes. Such processes will have the possibility of composing eavesdropped messages, decomposing messages and using cryptography whenever the appropriate keys are available. Below we present the six basic SPL spy processes, which refer to the standard Dolev-Yao [17] model. They are inspired from [14]. By choosing various specifications for the attacker (*i.e.*, various combinations of spy processes) one can restrict or increase its power of aggressiveness.

The following six spy processes may be divided into three groups depending on the action they perform. Thus, the first group deals with messages composition/decomposition: Spy_1 reads two messages from the network and issues to the network the message composed of the two read messages, while Spy_2 decomposes a message read on the network and issues its parts.

$$Spy_1 \stackrel{\text{df}}{=} in\ pat\ \{\Psi_1\}\{\Psi_1\}.\ in\ pat\{\Psi_2\}\{\Psi_2\}.\ out\ \{\Psi_1, \Psi_2\}$$
$$Spy_2 \stackrel{\text{df}}{=} in\ pat\ \{\Phi_1, \Phi_2\}\{\Phi_1, \Phi_2\}.\ out\ \{\Phi_1\}.\ out\ \{\Phi_2\}$$

The next group deals with encryption: Spy_3 encrypts a message with the public key of someone, and issues it to the network, while Spy_4 encrypts a message with a symmetric key, and issues it.

$$Spy_3 \stackrel{\text{df}}{=} in\ pat\ \{u, g\}\{\langle Pub, u, g\rangle\}.\ in\ pat\{\Psi\}\{\Psi\}.\ out\ \{\Psi\}_{\langle Pub,u,g\rangle}$$
$$Spy_4 \stackrel{\text{df}}{=} in\ pat\ \{u, g_1, g_2\}\{\langle Sym, u, g_1, g_2\rangle\}.\ in\ pat\{\Psi'\}\{\Psi'\}.\ out\ \{\Psi'\}_{\langle Sym,u,g_1,g_2\rangle}$$

The last group deals with decryption thanks to some obtained keys: Spy_5 decrypts a message with the private key of someone if it is available, and issues it to the network, while Spy_6 decrypts a message with a symmetric key if it is available, and issues it.

$$Spy_5 \stackrel{\text{df}}{=} in\ pat\ \{u, g\}\{\langle Priv, u, g\rangle\}.\ in\ pat\{\Phi\}\{\Phi\}\langle Pub, u, g\rangle.\ out\ \{\Phi\}$$
$$Spy_6 \stackrel{\text{df}}{=} in\ pat\ \{u, g_1, g_2\}\{\langle Sym, u, g_1, g_2\rangle\}.\ in\ pat\{\Phi'\}\{\Phi'\}_{\langle Sym,u,g_1,g_2\rangle}.\ out\ \{\Phi'\}$$

In case we need to manage more than one key of a kind per agent, the spy definitions take this information into account, for example,

$$Spy_3 \stackrel{\text{df}}{=} in\ pat\ \{u, g, i\}\{\langle Pub, u, g, i\rangle\}.\ in\ pat\{\Psi\}\{\Psi\}.\ out\ \{\Psi\}_{\langle Pub,u,g,i\rangle}$$

In this model, a general potentially very aggressive attacker is modeled by an infinite set of spy processes running in parallel.

2.3 A role based specification of the KC protocol

The roles in our modeling of the KC protocol comprise the following SPL process definitions, where $X, X', X'', \ldots$, are agent variables in V_G, $u, u', u'', \ldots$, are nonce variables in V_N, and $\Psi \in V_M$ is a message variable:

$$Init(X,Y,Z,u) \stackrel{\text{df}}{=} out\ new\ \{x\}\ \ \{X,Y,x\}.$$
$$\quad in\ pat\{y,z\}\ \ \{\{X,Y,\langle Sym,z,X,Y\rangle,x\}_{\langle Sym,u,X,Z\rangle},\{x\}_{\langle Sym,z,X,Y\rangle},y\}.$$
$$\quad out\ \ \{y\}_{\langle Sym,z,X,Y\rangle}\}$$

$$Serv(Z',u',v') \stackrel{\text{df}}{=} in\ pat\ \{X',Y',x'\}\ \ \{X',Y',x'\}.$$
$$\quad out\ \ new\{w'\}\ \ \{\{X',Y',\langle Sym,w',X',Y'\rangle,x'\}_{\langle Sym,u',X',Z'\rangle},$$
$$\{X',Y',\langle Sym,w',X',Y'\rangle,x'\}_{\langle Sym,v',Y',Z'\rangle}\}$$

$$Resp(Y'',Z'',u'') \stackrel{\text{df}}{=}$$
$$\quad in\ pat\{X'',y'',w'',\Psi\}\ \ \{\Psi,\{X'',Y'',\langle Sym,w'',X'',Y''\rangle,y''\}_{\langle Sym,u'',Y'',Z''\rangle}.$$
$$\quad out\ new\ \{x''\}\ \ \{\Psi,\{y''\}_{\langle Sym,w'',X'',Y''\rangle},x''\}.$$
$$\quad in\ \ \{x''\}_{\langle Sym,w'',X'',Y''\rangle}$$

A complete role based specification of the KC protocol comprising one initiator A, one server S and one responder B is given by

$$\begin{array}{l}(\{\ (Init(A,B,S,n_1),\{\langle Sym,n_1,A,S\rangle\}),\\ \quad (Serv(S,n_1,n_2),\{\langle Sym,n_1,A,S\rangle,\langle Sym,n_2,B,S\rangle\}),\\ \quad (Resp(B,S,n_2),\{\langle Sym,n_2,B,S\rangle\})\ \ \},\\ \\ \ \{\ Spy_1,Spy_2,Spy_4,Spy_6\ \ \},\\ \\ \ \mathcal{M}\cup\{\{A\},\{B\},\{S\}\}\ \ )\end{array}$$

where the roles $Init(X,Y,Z,u)$, $Serv(Z',u',v')$, $Resp(Y'',Z'',u'')$ and Spy_i are defined as above, and $\mathcal{M}$ is a set of residual messages. Here we use only the spy processes that make sense in the case of the KC protocol (for instance, Spy_3 deals with public key cryptography that is not involved in KC).

2.4 Different versions of the KC protocol

We will consider in the following two versions of the KC protocol and its environment. The first one will be used to check the intrinsic resistance of the protocol, and will start from an empty initial knowledge: $\mathcal{M}=\emptyset$. But since the essence of the KC protocol is to manage short term session keys, it could happen that a key becomes compromised after some time. This may happen, for example, due to more or less clever brute force attacks. So, we will consider a version where $\mathcal{M}$ contains initially such a compromised key. Another KC variant would be a system with various initiators and responders.

3 Petri net translation of a role based specification

3.1 The general picture

Given a role based specification, its Petri net semantics may be obtained compositionally using an algebra of high-level Petri nets, inspired from the S-nets

introduced in [9] and its optimised version defined in [10]. Basic (unmarked) nets in this algebra allow to directly represent the SPL prefixes and are one-transition nets, like those in Figures 1 and 2, where the places represent composition interfaces, which are either *control-flow* (entry, exit or internal) or *buffer* ones. The labeling of the last ones (that are devoted to asynchronous communications) may be either:

- SPL variable names (each such place storing the current value of the variable);
- κ_δ (storing information about encryption/decryption keys privately known by a role δ);
- or Ω (storing all the messages transferred through the public network).

These nets may be composed with various composition operations:

- sequentially ($N_1;N_2$, which means that the exit places of the first net and the entry places of the second net are combined);
- in parallel ($N_1 \| N_2$, which means that both nets are put side by side);
- or replicated ($!N = \|_{i\in\mathbb{N}} N$).

All these operations include the merging of buffer places having the same label in order to allow asynchronous communications.

Thus, if $(\delta(f_\delta), l_\delta)$ is an initialized role in a role based specification, the process expression that describes the behavior of δ allows to compositionally construct a corresponding unmarked S-net. Its initial marking is obtained by putting a (black) token into its entry places while the tokens in the buffer places are put accordingly to the context (f_δ, l_δ).

If *Spy* is a prototype role defining a kind of activity of the environment, we proceed similarly but implicitly assume that the parallel composition of all such roles may be replicated. Its finite structure representation is then obtained as an unmarked S-net, as defined in [10]. Its initial marking is defined by putting one token in each entry place and no token elsewhere. The complete Petri net representation of a role based specification is then a parallel composition of all the corresponding S-nets. Its initial marking includes the markings mentioned above and the tokens (messages) coming from the public context in the place Ω.

3.2 Translation scheme

Let us assume that

$$(\{(\delta_1(f_1), l_1), \ldots, (\delta_p(f_p), l_p)\}, \{s_1, \ldots, s_r\}, \mathit{Public\_Context})$$

is a role based specification, where each role δ_i is defined as

$$\delta_i(a_1, \ldots, a_{m_{\delta_i}}) \stackrel{\text{df}}{=} p_{\delta_i}.$$

Its translation into Petri nets is defined by the following phases, where the semantic function Snet will be detailed later on:

– *Phase 1.* Each role definition δ_i is translated first into an S-net $\mathsf{Snet}(\delta_i)$, then we construct a net R as the parallel composition $\mathsf{Snet}(\delta_1)\| \cdots \|\mathsf{Snet}(\delta_m)$, which merges in particular all the buffer places Ω and all the key places κ_{δ_i} for the same δ_i;
– *Phase 2.* Each role definition s_j (spy) is translated first into an optimized transformed S-net $\mathsf{Snet}(s_j)$. Then, we construct a net SPY as the parallel composition $\mathsf{Snet}(s_1)\| \cdots \|\mathsf{Snet}(s_r)$, which encodes every possible execution of the attackers. Note that this translation is similar for some class of specifications and may be performed only once;
– *Phase 3.* The nets R and SPY are composed in parallel $R\|\mathsf{SPY}$ and marked as follows:
  - the place Ω receives all the tokens (messages) from *Public_Context*;
  - each entry place receives a black token;
  - for each initialized role δ_i:
    * each key place κ_{δ_i} receives all the tokens from l_i;
    * each place a_j, for $0 \leq j \leq m_{\delta_i}$ receives the token $f_i(a_j)$.

3.3 Petri net semantics of roles

Each role definition $\delta(\cdots) \stackrel{\text{df}}{=} p_\delta$ may be translated compositionally into a corresponding S-net, as defined in [9], providing its high-level Petri net representation. S-nets, like other high-level Petri net models, carry the usual annotations on places (types, *i.e.*, sets of allowed tokens), arcs (multisets of net variables) and transitions (guards, *i.e.*, Boolean expressions that play the role of occurrence conditions). S-nets have also the feature of read-arcs, which are a Petri net specific device allowing transitions to check concurrently for the presence of tokens stored in their adjacent places [11]. Like other box-like models [6, 7], S-nets are also provided with a set of operations giving them an algebraic structure. In the version considered here, this is achieved through an additional labeling of places which may have a status (entry, exit, internal or buffer) used for net compositions.

The marking of an S-net associates to each place a multiset of values (tokens) from the type of the place and the transition rule is the same as for other high-level nets; namely, a transition t can be executed if the inscriptions of its input arcs can be mapped to values which are present in the input places of t and if the guard of t, $\gamma(t)$, is true under this mapping. The execution of t transforms the marking by removing values (accordingly to the mapping of arc inscriptions) from the input places of t and by depositing values into its output places. Read-arcs are only used to check the presence of tokens in the adjacent places but do not modify the marking. They are represented in figures as undirected arcs. Notice that given a transition t and a marking μ, there may be several possible executions (firings), corresponding to different mappings of variables in the inscriptions around t to values (called *bindings* and denoted σ).

More precisely, we distinguish the following sets of tokens as types for the places in S-nets.

– $\{\bullet\}$ for the entry, internal and exit places;
– Msg for the buffer place labeled Ω intended to gather all the messages present in the network, as well as for all buffer places labeled with SPL message variables (from V_M), even if the latter are intended to carry at most one message. For implementation or efficiency reasons, the set Msg may be restricted, for instance to a given maximal message length;
– $N \cup G \cup K \cup \mathsf{Msg}$ for all the other buffer places.

The S-net transitions always have at least one input and at least one output control-flow place; they may be of two sorts, corresponding to the two basic SPL actions 'in' and 'out'. Arcs are inscribed by sets of net variables (denoted $a, b, c, d \ldots$). In order to avoid too complex (hence difficult to read) transition guards in some figures, we use a shorthand allowing to put complex terms as arc inscriptions (which has no consequences on the behavior of the net): we may thus replace an arc annotation c and a guard $c = f(a, b, d)$ by the arc annotation $f(a, b, d)$ and the true (or empty) guard.

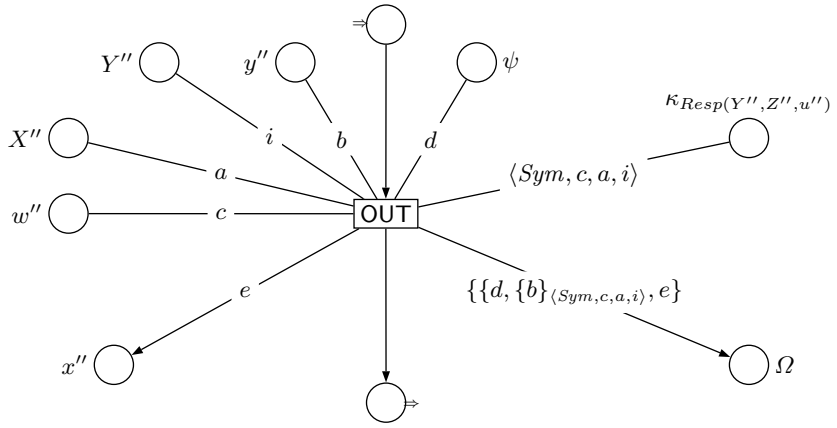

Fig. 1. The S-net corresponding to $out\, new\, \{x''\}\{\Psi, \{y''\}_{\langle Sym,w'',X'',Y''\rangle}, x''\}$ in the role $Resp(Y'', Z'', u'')$.

By analogy with the two basic SPL actions, there are two basic S-nets. Each of them has a single transition, inscribed IN or OUT, one entry and one exit place of type $\{\bullet\}$, one buffer place labeled Ω of type Msg, a unique key place κ_δ where δ is the role in which the modeled action occurs, and some other buffer places, as shown in Figures 1 and 2.

For instance, the SPL output action

$$out\ new\ \{x''\}\{\Psi, \{y''\}_{\langle Sym,w'',X'',Y''\rangle}, x''\}$$

in the role $Resp(Y'', Z'', u'')$ gives rise to the S-net represented in Figure 1. This net may be executed if there is a token $\bullet$ in the entry place (denoted by an incoming $\Rightarrow$), and if the buffer places labeled w'', y'', X'', Y'', Z'', Ψ and $\kappa_{Resp(Y'',Z'',u'')}$, are marked (*i.e.*, contain values). At the firing of the transition under a binding σ, a nonce[1] is generated and put in place x''; the tokens existing in places w'', y'', X'', Y'', Z'', Ψ and $\kappa_{Resp(Y'',Z'',u'')}$ are read (but not removed); the token $\bullet$ is transferred from the entry place to the exit place (denoted by an outgoing $\Rightarrow$); and the whole message $\{\sigma(d), \{\sigma(b)\}_{\langle Sym, \sigma(c), \sigma(a), \sigma(i)\rangle}, \sigma(e)\}$ is put in the message buffer Ω.

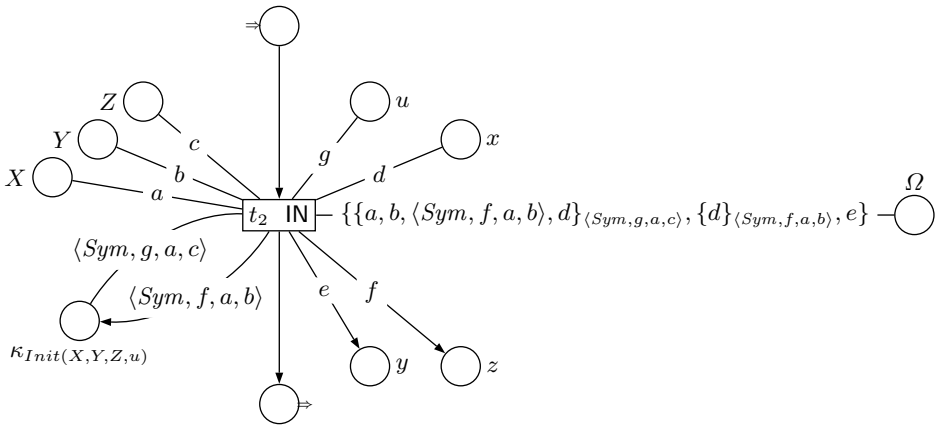

Fig. 2. The S-net for the 'in' action in the role $Init(X, Y, Z, u)$.

The elementary SPL input action

$$in\ pat\ \{y, z\}\{\{X, Y, \langle Sym, z, X, Y\rangle, x\}_{\langle Sym,u,X,Z\rangle}, \{x\}_{\langle Sym,z,X,Y\rangle}, y\}$$

of the role $Init(X, Y, Z; u)$ is obtained analogously and represented in Figure 2. The message $\{\{X, Y, \langle Sym, z, X, Y\rangle, x\}_{\langle Sym,u,X,Z\rangle}, \{x\}_{\langle Sym,z,X,Y\rangle}, y\}$ has local nonce variables y, z and u, global nonce variables x, X, Y, Z and no message variable. At the firing of the transition, a message from Ω with the adequate pattern is read, such that the existing values of x, X, Y and Z in the corresponding places are checked (through the read-arcs) to be equal to the corresponding elements of the message, and the value of the variables y and z are decoded from the message and put in the corresponding places.

The S-net representing a role δ is defined compositionally using S-net composition operations and is denoted $\mathsf{Snet}(\delta)$.

We can observe, that the place Ω is part of each basic net and, through merging, thus becomes global to each composed net, like in Figures 5, 6 and 7.

[1] We assume that σ is such that $\sigma(e)$ is fresh. This can be implemented by using a place initially marked by a set of nonce values that are consumed when a fresh one has to be generated.

3.4 S-net semantics of the environment

The most general environment of a protocol is the most aggressive attacker, thus an unbounded number of all sorts of spies. Its role SPY can be described as a replication of all involved spies in parallel (for KC we have $J = \{1, 2, 4, 6\}$, 3 and 4 being irrelevant in this context).

$$\mathsf{SPY} = \ !\ (\|_{i \in J} Spy_i) \quad \text{or equivalently} \quad \mathsf{SPY} = \ \|_{i \in J}\ !\ Spy_i.$$

The associated net Snet(SPY) is that obtained by the infinite parallel composition of the S-nets obtained via the standard translation of the various Spy_i's.

For instance, Figure 3 shows the standard translation of our first spy.

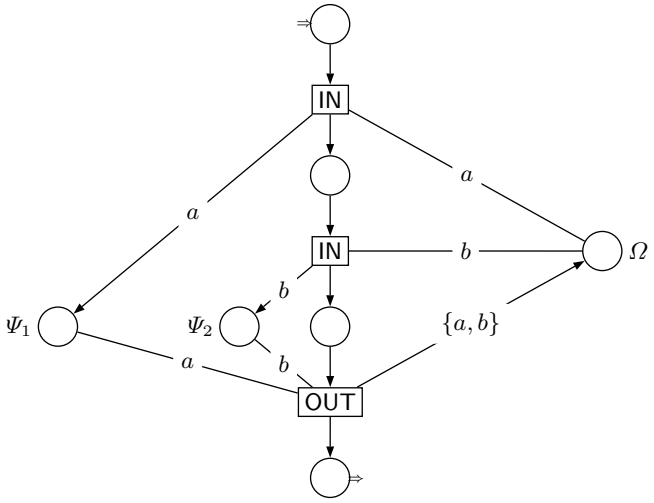

Fig. 3. One copy of the S-net of Spy_1.

This representation and translation of the environment presents some difficulties with respect to verification: the process SPY representing all potential attackers is infinite, as well as Snet(SPY). A way to cope with this problem is to propose net transformations which do not essentially modify the net behaviour but lead to finite net and marking graph. This is possible, as shown in [10], by proposing solutions to the following three problems, given by increasing difficulty:

1. The composed messages do not have a bounded length.
2. The net is infinite (even for a bounded number of agents) because of replication in SPY.
3. The multiplicity of tokens in place Ω is not bounded.

The first and third ones induce infinite paths in the marking graph, the second one infinite edge degrees. We just like to quote here the adopted solutions which ensure finiteness.

1. The length of composed messages will be bounded by a (usually very small) constant, which corresponds to the maximum length of valid messages in the considered protocol; hence, the type Msg of buffer places will be a bit restricted, but the messages not representable are useless for the protocol.
2. Replication will be replaced by iteration, in the form of a loop. Thus we have only to consider $|I|$ transformed Spy_i-nets which are cyclic, and where at each cycle the buffer places which should initially be empty are emptied by a "flush" transition. The resulting behaviour is not exactly the same as with a true replication, since no two copies of a same spy may be simultaneously in the middle of their work; but this has no impact on the messages they can produce in Ω.
3. The place Ω will contain a multi-set with a very small bound of multiplicities. This can be achieved by adding a place am_i (for access-memory) to each Spy_i net, containing a local copy of Ω : its type is organized as a list without repetition, ensuring (via the guard of the upper transition) that no multiple copies are put in the global place Ω by this Spy. (Identical copies may be still be produced by different spies, but there is a small number of them.) Additional places (see ss_1 in Figure 4) allow to force the next transitions to use the same values as the first one when necessary.

This way, we obtain an optimized transformed spy-net for each Spy_i, called $\mathsf{Snet}(Spy_i)$, as illustrated (for $i = 1$) in Figure 4. Via an adequate equivalence

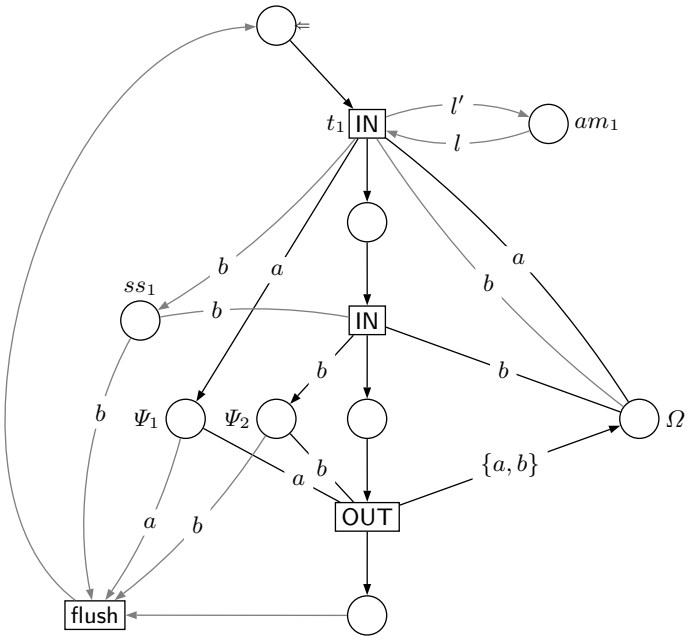

Fig. 4. The net $\mathsf{Snet}(Spy_1)$, where the guard of t_1 is $((a.b) \notin l) \wedge l' = l.(a, b)$.

relation on these nets, it has been shown in [10], that these transformations preserve the (interesting part of the) original behaviour: $\mathsf{Snet}(Spy_i) \equiv\ !\ Spy_i$.

By taking $\mathsf{Snet}(\mathsf{SPY}) = \|_{i\in I}\ \mathsf{Snet}(Spy_i)$ we have a finite net representation of the complete SPY role.

4 Petri net model of the KC protocol and its verification

4.1 S-net semantics of KC agents

There are three agents involved in the KC protocol: the initiator, the server and the responder. The first one is represented by the role $Init(X, Y, Z, u)$, where X denotes the name of the agent itself, Y the name of the responder and Z the name of the server involved. The variable u denotes the value of the symmetrical key that the initiator shares with the server. So we get for the initiator the S-net depicted in Figure 5.

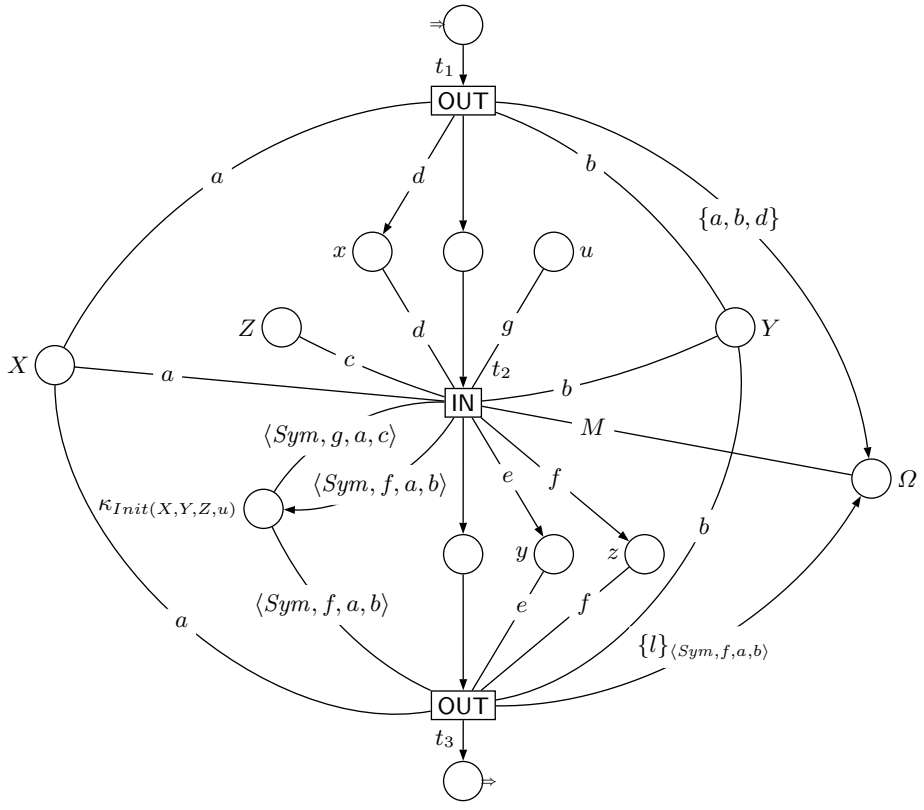

Fig. 5. The S-net for $Init(X, Y, Z, u)$, where the guard of t_2 is the equality $M = \{\{a, b, \langle Sym, f, a, b\rangle, d\}_{\langle Sym,g,a,c\rangle}, \{d\}_{\langle Sym,f,a,b\rangle}, e\}$.

The second one is the server, represented by the role $Serv(Z', u', v')$, whose free variables are Z', u', v', denoting respectively the name of the server itself, and two key values that the server will distribute on receiving the first message from the initiator. The corresponding S-net is depicted in Figure 6.

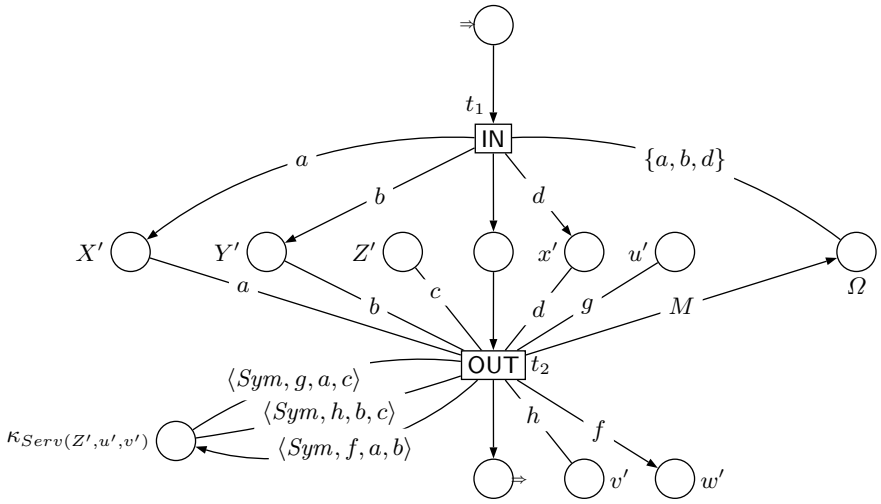

Fig. 6. The S-net for $Serv(Z', u', v')$, where the guard of t_2 is $M = \{\{a, b, \langle Sym, f, a, b\rangle, d\}_{\langle Sym,g,a,c\rangle}, \{a, b, \langle Sym, f, a, b\rangle, d\}_{\langle Sym,h,b,c\rangle}\}$.

Finally, the responder corresponds to the role $Resp(Y'', Z'', u'')$, where Y'' represents the name of the responder itself, Z'' to the server and u'' is the value of the key that is shared between the Z'' and Y''. The corresponding S-net is shown in Figure 7.

The S-net of the complete protocol is made of the agents and the environment Snet(SPY), as explained above.

4.2 Automated verification of KC

The next step of our approach is the automated verification of the authentication property of KC using model-checking techniques. We used for that the Helena analyzer [19], which is a general purpose high-level Petri net model-checker.

Starting from the role based specification, as described above, we implemented in Helena the S-nets of the roles and of the environment.

We considered first the intrinsic resistance of the KC protocol, *i.e.*, the variant where the public context $\mathcal{M}$ is empty. In order to test the resistance of the protocol in such an environment, we used a complete protocol with two agents A, B and a server S, with the environment formed of Spy_1, Spy_2, Spy_4 and Spy_6.

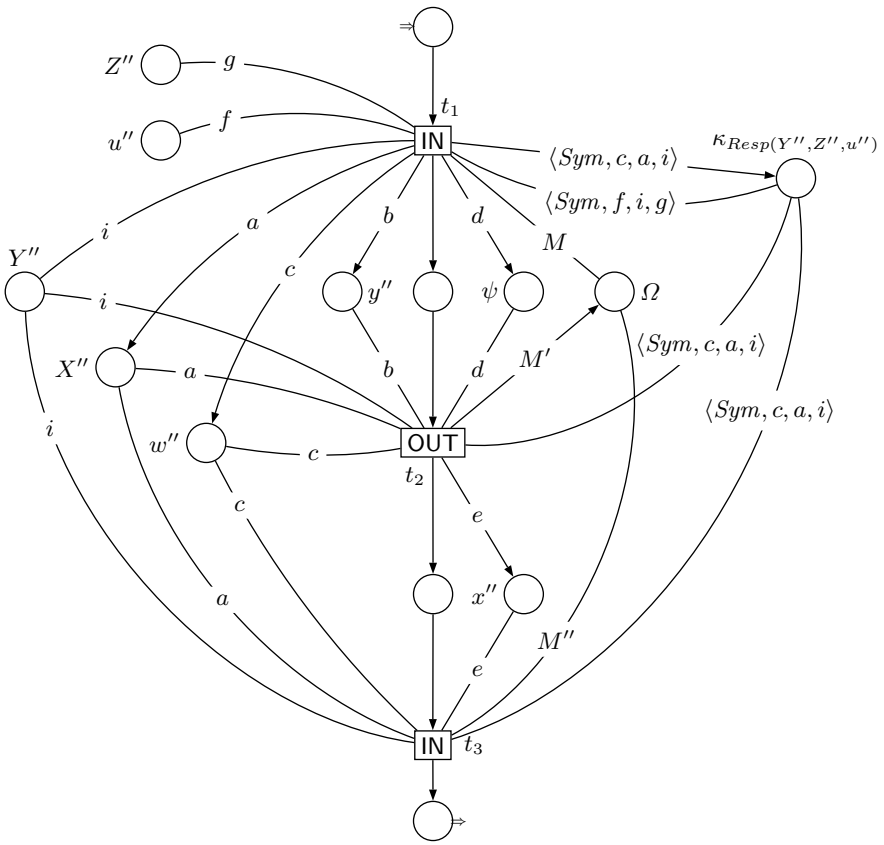

Fig. 7. The S-net for $Resp(Y'', Z'', u'')$, where the guard of t_1 is $M = \{d, \{a, i, \langle Sym, c, a, i\rangle, b\}_{\langle Sym,f,i,g\rangle}\}$, the guard of t_2 is $M' = \{\{d, \{b\}_{\langle Sym,c,a,i\rangle}, e\}$, and the guard of t_3 is $M'' = \{e\}_{\langle Sym,c,a,i\rangle}$.

In the KC protocol, B authenticates A when it receives the last message $\{n\}_{K_{ab}}$ from A, while A authenticates B on receiving the message

$$\{A, B, K_{ab}, m\}_{K_{as}}, \{m\}_{K_{ab}}, n.$$

That implies B authenticates after A does. In the net vocabulary, agents A and B mutually authenticate when the exit places of $\mathsf{Snet}(Init(A, B, S, n_1))$ and $\mathsf{Snet}(Resp(B, S, n_2))$ are marked. Therefore, a first property to check is that it is not possible to reach a marking where the exit place of $\mathsf{Snet}(Resp(B, S, n_2))$ is marked while the exit place of $\mathsf{Snet}(Init(A, B, S, n_1))$ is not. This verification was successfully performed: no such violating marking is reached when the initial knowledge is empty.

The second analysis was to check if the protocol still resists when the public context is not initially empty. It means that agents A and B have already used the KC protocol. So, some of the messages exchanged during previous sessions

are part of the public context and we can assume also that for some reason a previous key session is compromised and was made public by some other (brute force, for instance) attacker. Therefore, it means in terms of the net semantics, that the place Ω contains the messages $\{\langle Sym, n_3, B, S\rangle\}$ and

$$\{\{A, B, \langle Sym, n_3, A, B\rangle, a\}_{\langle Sym, n_1, A, Z\rangle}, \{A, B, \langle Sym, n_3, A, B\rangle, a\}_{\langle Sym, n_2, B, Z\rangle}\}$$

together with the messages containing the names of the agents involved in the protocol $\{A\}, \{B\}, \{S\}$, as usual.

In this case, violating markings are the same as previously: the exit place of $\mathsf{Snet}(Resp(B, S, n_2))$ is marked and the exit place of $\mathsf{Snet}(Init(A, B, S, n_1))$ is not. In that case we found that the violating marking was reached and therefore this implies that the authentication property is violated. This corresponds to the fact that the spies succeeded in convincing B that a successful new session has been started with A, while this is not the case.

It took 29s using the Helena model-checker to compile and analyse the entire system in both cases on an Intel® Core$^{\text{TM}}$Duo 2.33GHz.

5 Conclusion and future work

Security protocols are a very delicate field. Since they imply concurrent agents communicating through a generally public and insecure medium (allowing attackers to add, suppress and forge messages), they are prone to numerous forms of weaknesses, difficult to track and discover. And indeed, it is not uncommon that a protocol seems (sometimes "trivially") correct and is used for years before a flaw is discovered. A famous example of this kind is for instance the Needham-Schroeder protocol [28], broken 17 years after its introduction [23, 24].

Hence it appears necessary to use more formal and systematic ways to assess such protocols [25], like for instance [2, 22, 13, 8]. Several approaches, like theorem provers or model-checking, were applied and sometimes offered in integrated environments like CAPSL [15] or AVISPA [1]. Also, type-checking [20, 2] has been recently used, which like model-checking has the advantage to be completely automatic, but since security violations are defined in terms of type inconsistencies, the security property to be proved has to be considered when the specification is being written. Among various model-checking approaches, one may quote for instance Murφ [16, 27] (based on condition/action rules and state space exploration), Casper [18] (generating CSP specification for the FDR model-checker) or Athena [32, 33] (based on the strand space model, specialised logic and model-checker). They are mostly not process but message transfer oriented, often consider intruder implicitly and do not always explicitly express local and global knowledge.

Since Petri nets were especially introduced to cope with concurrency problems (the interleaving semantics is often enough to do so, but true concurrent semantics are also available) and to model accesses to common resources (through places of low-level or high-level nets), and since effective tools are now available to analyse and model-check them, we felt appealing to use this approach, like [29, 5].

We thus introduced, and illustrated on a concrete case study (the Kao-Chow authentication protocol), a role based specification framework devoted to the specification and verification of properties of security protocols. It is composed of three uniformly defined parts describing the behaviour of the roles, the behaviour of the environment (representing a potentially aggressive attacker), and the global and local knowledges (about agent names, messages, or public, private and shared keys) the roles can have.

Unlike usual security protocol notations do, that is similar to what we used in the introduction, the advantage of a role based specification is that it is fully explicit and formal. The roles and the environment are expressed using SPL processes provided with their private contexts, and a definition of a global context. These SPL descriptions are then compositionally translated into high-level Petri nets, while the context (depending on the studied property) is used to generate the corresponding initial marking. An immediate advantage of the method is that the obtained Petri net model can be analysed using standard model-checking or simulation tools.

Compared to our previous approaches [9, 10], the present paper introduce novel features for the treatment of symmetric keys as well as asymmetric ones, together with a uniform way to analyse various forms of attack contexts.

We already started to use this framework to analyse other kinds of protocols, mixing both symmetric and asymmetric keys, and allowing other kinds of attack contexts. This should be the subject of forthcoming papers.

References

1. Armando, A. et al.: The AVISPA tool for the automated validation of internet security protocols and applications, CAV'2005, LNCS 3576, Springer (2005)
2. Abadi, M., Blanchet, B.: Analyzing security protocols with secrecy types and logic programs. Journal of the ACM, 52(1), (2005)
3. Abadi, M., Gordon, A.: A calculus for cryptographic protocols. The Spi calculus. ACM Conference on Computers and Communication Security, ACM Press, (1997)
4. Abadi, M., Rogaway, P.: Reconciling two views of cryptography (the computational soundness of formal encryption). Journal of Cryptology, 15(2) (2002)
5. Al-Azzoni, I., Down, D.G., Khedri, R.: Modelling and verification of cryptographic protocols using coloured Petri nets and Design/CPN. Nordic Journal of Computing, 12(3) (2005)
6. Best, E., Devillers, R., Koutny, M.: Petri Net Algebra. EATCS Monographs on TCS, Springer, ISBN 3-540-67398-9 (2001)
7. Best, E., Frączak, W. Hopkins, R. P., Klaudel, H., Pelz, E.: M-nets: an algebra of high level Petri nets, with an application to the semantics of concurrent programming languages. Acta Informatica, 35, Springer, (1998)
8. Blanchet, B.: A computationally sound mechanized prover for security protocols. In IEEE Symposium on Security and Privacy, (2006)
9. Bouroulet, R., Klaudel, H., Pelz, E.: A semantics of Security Protocol Language (SPL) using a class of composable high-level Petri nets. ACSD'04, IEEE Computer Society, (2004)

10. Bouroulet, R., Klaudel, H., Pelz, E.: Modelling and verification of authentication using enhanced net semantics of SPL (Security Protocol Language). ACSD'06, IEEE Computer Society (2006)
11. Christensen, S., Hansen, N.D.: Coloured Petri Nets Extended with Place Capacities, Test Arcs and Inhibitor Arcs. ATPN'93, LNCS vol. 691, Springer (1993)
12. Clark, J., Jacob, J.: A survey of authentication protocol literature : Version 1.0., `http://www.cs.york.ac.uk/jac/papers/drareview.ps.gz` (1997)
13. Cortier, V., Warinschi, B.: Computationally sound, automated proofs for security protocols. ESOP'05, LNCS vol. 3444, Springer (2005)
14. Crazzolara, F., Winskel, G.: Events in security protocols. ACM Conf on Computer and Communications Security, ACM Press (2001)
15. Denker, G., Millen, J.: CAPSL Integrated Protocol Environment. DISCEX'00, IEEE Computer Society (2000)
16. Dill, D. L., Drexler, A.J., Hu, A. J., Han Yang, C.: Protocol Verification as a Hardware Design Aid. IEEE International Conference on Computer Design: VLSI in Computers and Processors (1992)
17. Dolev, D., Yao, A.C.: On the security of public key protocols. IEEE, Transactions on Information Theory IT-29 12 (1983)
18. Donovan, B., Norris, P., Lowe, G.: Analyzing a library of security protocols using Casper and FDR. Workshop on Formal Methods and Security Protocols (1999)
19. Evangelista, S.: High Level Petri Nets Analysis with Helena. ICATPN'05, LNCS vol. 3536, Springer, `http://helena.cnam.fr/` (2005)
20. Gordon, A., Jeffrey, A.: Authenticity by Typing for Security Protocols. IEEE Computer Security Foundations Workshop, IEEE Computer Society Press (2001)
21. Kao, I.-L., Chow, R.,: An efficient and secure authentication protocol using uncertified keys. Operating Systems Review, ACM Press, 29(3) (1995)
22. Kremer, S., Raskin, J.-F.: A Game-Based Verification of Non-Repudiation and Fair Exchange Protocols. Journal of Computer Security 11(3) (2003)
23. Lowe, G.: An attack on the Needham-Schroeder public key authentication protocol. Information Processing Letters, 56(3) (1995)
24. Lowe, G.: Breaking and fixing the Needham-Schroeder public-key protocol using (FDR). TACAS'96, LNCS vol. 1055, Springer (1996)
25. Meadows, C.: Formal Methods for Cryptographic Protocol Analysis: Emerging Issues and Trends. IEEE Journal on Selected Areas in Communication, Vol. 21(1) (2003)
26. Milner, R.: Communicating and mobile systems: The π-calculus, Cambridge University Press (1999)
27. Mitchell, J. C., Mitchell, M., Stern, U.: Automated Analysis of Cryptographic Protocols Using Murφ. IEEE Symposium on Security and Privacy, IEEE Computer Society Press, (1997)
28. Needham, R.M., Schroeder, M.D.: Using Encrypton for Authentication in Large Networks of Computers. Comm. of the ACM, 21(12) (1978)
29. Nieh, B.B., Tavares, S.E.: Modelling and Analyzing Cryptographic Protocols Using Petri Nets, AUSCRYPT'92, LNCS vol. 718, Springer (1993)
30. Rivest, R.L. Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystem, Comm. of the ACM, 21(2) (1978)
31. Schneier, B.: Applied Cryptography. Wiley, ISBN 0-471-11709-7 (1996)
32. Thayer, F., Herzog, J.C., Guttman, J.D.: Strand Spaces: Why is a Security Protocol Correct? IEEE Symposium on Security and Privacy (1998).
33. Song, D.: Athena: A new efficient automatic checker for security protocol analysis, CSFW'99. IEEE Computer Society Press (1999)