A semantics of Security Protocol Language (SPL) using a class of composable high-level Petri nets

Bouroulet, R. and Klaudel, H. and Pelz, E.

A semantics of Security Protocol Language (SPL)
using a class of composable high-level Petri nets

Roland Bouroulet
LACL, Université Paris 12,
61 av. du gén. de Gaulle, F-94010 Créteil.
bourole@univ-paris12.fr

Hanna Klaudel
LaMI, Université d’Evry,
bd. François Mitterrand, F-91000 Evry.
klaudel@lami.univ-evry.fr

Elisabeth Pelz
LACL, Université Paris 12,
61 av. du gén. de Gaulle, F-94010 Créteil.
pelz@univ-paris12.fr

Abstract

This paper aims at introducing a Petri net semantics of security protocols allowing to study their properties formally. This is obtained by means of an economic but expressive class of composable high-level Petri nets, called S-nets, inspired from works about the relationship between Petri nets and process algebras. S-nets are applied then to give a compositional high-level Petri net semantics to SPL. We will employ the Needham-Schröder protocol to illustrate how this semantics can be used in order to establish the violation of the authentication property.

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 public key infrastructure, e.g., [RSA78], where public keys are available to all participants or agents, while a private key is secret and only known by its owner. An agent needs its private key to decrypt received messages which are encoded using its public key. Examples of such protocols are, for instance, the Needham-Schröder authentication protocol (NS) [NS78], and its correction by Lowe (NSL) [L96]. These protocols have two roles: one for the initiator and one for the responder. They make use of nonces, which are newly created names (or values), which one can interpret as fresh session keys, the agents may use for secure communication after agreement, i.e., at the end of the protocol if they are sure about the secrecy of their knowledge about keys.

The NS protocol between $A$ (initiator) and $B$ (responder) may be described as follows:

1. $A \rightarrow B : \{m, A\}_{\text{Pub}(B)}$
   The initiator agent $A$ sends to $B$ a nonce $m$ together with its own agent name $A$ encrypted with $B$’s public key. Once received, $B$ decrypts the message with its private key.

2. $B \rightarrow A : \{m, n\}_{\text{Pub}(A)}$
   Then, $B$ sends to $A$ a message made up of the received nonce $m$ and another nonce $n$, encrypted with $A$’s public key, which $A$ should decrypt after reception.

3. $A \rightarrow B : \{n\}_{\text{Pub}(B)}$
   Finally, $A$ sends $n$ back to $B$, encrypted with $B$’s public key, in order to certify that the communication is well established.

Most of the time, the protocols are described in such an intuitive way, see also [S96], as it makes it difficult to think about suitable properties like, for instance, secrecy or authentication. Formal methods may certainly help, as it was the case for the NS protocol, which has been shown, by Lowe [L96], to be prone to a “middleman” attack, using model-checking techniques. Its corrected version, called NSL [L96] fixed the error by modifying the responder part which has to include the name of the sender: in NSL, the second NS message becomes $B \rightarrow A : \{m, n, B\}_{\text{Pub}(A)}$.

In order to specify more precisely security protocols, formal description languages have been proposed, as the Spy Calculus in [AG97] or the Security Protocol Language, SPL in [CW01] (which is the starting point of our approach).
SPL is an economical process language inspired from a process algebra like asynchronous $\pi$-Calculus [M99]: each agent (or 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. For instance, in the SPL representation of the NS protocol, the fact that $A$ sends a message (made up of a nonce and the agent’s name) to $B$ may be expressed by the out-action $\text{out new } \{y\} \{y, A\} \text{Pub}(B)$, which means that the variable $y$ will be instantiated by a nonce and the message $\{y, A\}$, containing this nonce and the name $A$, encrypted with the public key of $B$, will be output to the network. On the other hand, the fact that $B$ awaits this kind of message may be expressed in SPL with the in-action $\text{in pat } \{x, Z\} \{x, Z\} \text{Pub}(B)$.

SPL was presented in [CW01] together with a semantics in terms of sets of events inspired by term rewriting. The denotation of a process is a set of events, and as each event specifies a set of pre and postconditions, this denotation can be viewed as a (low-level) Petri net; more precisely, a restricted version of a contextual net. Each particular execution of an SPL term gives rise to a Petri net, where places are labelled with input/output actions and messages, and where all variable substitutions and messages are anticipated. So, an SPL term corresponding, for instance, to a description of a protocol, leads to a large (potentially infinite) set of low-level Petri nets.

We aim at proposing a translation of SPL specifications into Petri nets in a different manner which preserves the structure of the SPL specification over possible evolutions, and the changes of the state are represented explicitly (as net markings). To cope with this, we use in this paper a restricted class of high-level Petri nets, called S-nets and dedicated to security protocols, which are provided with a composition mechanism particularly useful for a compositional translation from SPL. In this approach, inspired from the work on the relationship between process algebras, languages and Petri nets [BDK01, BFH98, KP99], each SPL input/output action will give rise to a basic S-net and the semantics of an SPL term will be obtained by composing all these nets together following the structure of the term and using S-net composition operations. Unlike the contextual net approach, where only ordinary "black" tokens are allowed, the places of an S-net may carry tokens which are messages, values (nonces) or agent names. This means that the tokens which flow through an S-net are instances of messages, nonces or agent names. They are calculated, assigned, transmitted and exchanged dynamically during the execution of the net. Thus, for a given SPL term, all its possible executions may be generated as evolutions of the corresponding (unique) S-net.

By means of an example taken from [CW01], we illustrate how our high-level Petri-net semantics can be used to prove security properties. In particular, we will show that the authentication property is violated for the NS protocol and can be established for the NSL one.

2 The Security Protocol Language SPL

We assume countable infinite disjoint sets of names of nonces $N = \{n, n', n'', \ldots\}$, of names of agents $G = \{A, B, \ldots\}$ and of indices for processes $I \supseteq \{i, \ldots\}$. We distinguish three disjoint sets of variables: for nonces $V_N = \{u, v, w, x, y, z, \ldots\}$, for agents $V_G = \{X, Y, Z, \ldots\}$ and for messages $V_M = \{\Psi, \Psi', \Psi_1, \ldots\}$. We use the vector notation $\vec{x}$ which abbreviates some possibly empty list $x_1, \ldots, x_n$, usually written $\{x_1, \ldots, x_n\}$.

The syntax of SPL is given by the following grammar, defining name expressions $e$, key expressions $k$, messages $M$ and processes $p$:

$$
eq e ::= n, A, \ldots \ y, Y, \ldots$

$$
k ::= \text{Pub}(e) | \text{Priv}(e) | \text{Key}(e_1, e_2)

$$
$$
M ::= e | k | M_1 \{e\} | \{\Psi\}

$$
$$
p ::= \text{out new } \vec{y} M.p | \text{in pat } \vec{y} M.p | \{e\} \in I p_i

$$

(a) We take $f_v(M)$, the free variables of a message $M$, to be the set of variables which appear in $M$. Then the free variables of process terms are defined as follows:

$$
\text{f_v(out new } \vec{y} M.p) = (f_v(p) \cup f_v(M)) \setminus \vec{y},$$

$$
\text{f_v(in pat } \vec{y} M.p) = (f_v(p) \cup f_v(M)) \setminus \vec{y}, \vec{y},$$

$$
\text{f_v( }\{e\} \in I p_i = \bigcup_{i \in I} f_v(p_i).

$$

As usual, a process or message is closed if it does not contain free variables. Also, we use standard notations for substitutions (bindings) of variables to names or closed messages; we will only be concerned with such substitutions.

(b) $\text{Pub}(e), \text{Priv}(e), \text{Key}(e_1, e_2)$ are used for the public key of $e$, the private key of $e$, and the symmetric keys shared by $e_1$ and $e_2$. A message may be a name, a key, a composition of two messages $(M_1, M_2)$ or an encrypted message $M$ using a key $k$, which is written $\{M\}_k$. By convention, messages are always enclosed in brackets $\{\}$.

(c) In the process out new $\vec{y} M.p$, the out-action prefixing $p$ chooses fresh distinct names $\vec{n} = \{n_1, \ldots, n_l\}$, binds them to variables $\vec{y} = \{y_1, \ldots, y_l\}$ and sends the message $M[\vec{n}/\vec{y}]$ out to the network, then the process resumes as $p[\vec{n}/\vec{y}]$. The new construct, like in [CW01], insures freshness of values in $\vec{n}$. Notice that communications are considered as asynchronous which means that output actions do not need to await input. By convention, we may simply omit the keyword new in an out-action if the list of "new" variables is empty.
(d) In the process in pat \( \vec{y} \vec{\Psi} M.p \), the in-action prefixing \( p \) awaits an input that matches the pattern \( M \) for some binding of the pattern variables \( \vec{y} \) and \( \vec{\Psi} \), then the process resumes as \( p \) under this binding, i.e., as \( p[I/\vec{y}, N/\vec{\Psi}] \). It is required that all the pattern variables \( \vec{y} = \{y_1, y_2, \ldots, y_l\} \) and \( \vec{\Psi} = \{\Psi_1, \Psi_2, \ldots, \Psi_j\} \) appear in \( M \).

(e) The process \( \|_{i \in I} p_i \) is the parallel composition of all processes \( p_i \) with \( I \subseteq I \). The particular case, where \( I \) is empty defines the process \( \text{nul} = \|_{i \in \emptyset} p_i \); by convention, we omit it almost always in examples. Replication of a process \( p \), denoted \( ! p \), stands for an infinite composition \( \|_{\omega} p \).

Without loss of generality, we will only consider processes which are well-formed, i.e., where there is no confusion in the usages of variables (for instance, none variable appears in two different new constructs). The well-formed processes can be defined by induction. For instance, the SPL process

\[
\text{out new}\{x, z\} \{x, y, z\} \text{Pub}(Z), \\
\text{out new}\{x, y\} \{x, y, z\} \text{Pub}(Z)
\]

is not well-formed because \( x \) appears in two new constructs, and also because the occurrence of \( y \) in the first out-action is free while all the occurrences of \( y \) in the second out-action are bound by the new construct (so, they do not represent the same thing). It is easy to rename coherently the bound variables in order to avoid clashes; a well-formed (equivalent) version of this process is, for instance,

\[
\text{out new}\{x, z\} \{x, y, z\} \text{Pub}(Z), \\
\text{out new}\{x', y'\} \{x', y', z\} \text{Pub}(Z).
\]

Notice that, for the second out-action, the variable \( z \) in the message \( \{x', y', z\} \text{Pub}(Z) \) is bound by the outer new \( \{\ldots z\} \) in the preceding out-action and so has to be that value sent out earlier.

Let in pat \( \vec{y} \vec{\Psi} M \) be an arbitrary SPL in-action, where \( f v(M) = \vec{y} \cup \vec{x} \cup \vec{\Psi} \cup \vec{\Phi} \), with \( \vec{y} \) and \( \vec{x} \) being disjoint sets of name variables, and \( \vec{\Psi} \) and \( \vec{\Phi} \) disjoint sets of message variables. We will say that

- \( \vec{y} \) are the local name variables of the in-action,
- \( \vec{\Psi} \) are its local message variables,
- \( \vec{x} \) are its global name variables and
- \( \vec{\Phi} \) are its global message variables.

and denote sometimes \( M \) explicitly by \( M(\vec{y}, \vec{\Psi}; \vec{x}, \vec{\Phi}) \). Local and global variables of an arbitrary SPL out-action are well-formed, and in particular, there is no local message variable. Similarly, we use in that case the notation \( M(\vec{y}; \vec{x}, \vec{\Phi}) \).

2.1 The protocols as SPL processes

The SPL terms for the initiator and the responder of the NS protocol are as follows:

\[
\text{Init}(A, B) = \text{out new}\{y\} \{y, A\} \text{Pub}(B), \\
in \text{pat}\{z\} \{y, z\} \text{Pub}(A), \\
\text{out}\{z\} \text{Pub}(B)
\]

\[
\text{Resp}(B) = \text{in}\{x, Z\} \{x, Z\} \text{Pub}(B), \\
\text{out new}\{v\} \{x, v\} \text{Pub}(Z), \\
in \text{pat}\{v\} \{v\} \text{Pub}(B)
\]

By putting \( \text{Init}(A, B) \) and \( \text{Resp}(B) \) in parallel, we obtain the simple NS protocol, and its execution (or run) corresponds to the following behaviour:

<table>
<thead>
<tr>
<th>( \text{Init}(A, B) )</th>
<th>( \text{Resp}(B) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{out new}{y} {y, A} \text{Pub}(B)</td>
<td>\text{in}{x, Z} {x, Z} \text{Pub}(B)</td>
</tr>
</tbody>
</table>

The initiator binds a nonce to the variable \( y \) and sends a message composed of this nonce and its name \( A \) to the agent \( B \). \( B \) awaits a message with two parameters, and binds the received values to the pattern variables \( x \) and \( Z \) through \( \sigma \).

\[
\text{B binds a nonce to the variable} v \text{ and sends back to the agent} \sigma(Z) \text{ a message composed of this nonce and the other received value,} \sigma(x) \text{.} A \text{ waits a message with a value to be used to bind the pattern variable} z \text{ through} \sigma' \text{, and with a confirmation of its previous message, i.e., with the nonce it sent before.}
\]
Although only two agents in their respective roles are described, the protocol is a shorthand for a wider situation of a network of distributed agents where each one is able to have a lot of concurrent sessions as both initiator and responder. The honest ones will stick to the protocol, but there are also attackers [S96] which might dissemble and pretend to be an other agent, taking advantage of any leaked keys it possesses in decrypting, and preparing falsified messages. The authentication property is violated if an attacker is successful: Lowe [L96] showed that the NS protocol is prone to a “middle-man” attack, if the name B is not included in the second message. Attackers may be described using “spy” processes they can use: Such processes will have the possibility of composing eavesdropped messages, decomposing messages and using cryptography whenever the appropriate keys are available. Below we present four basic SPL $Spy$ processes, taken from [CW01]. By choosing various specifications for the attacker (i.e., various combinations of $Spy$ processes) one can restrict or augment its power of aggressiveness. The replication of parallel composition of $Spy$ processes defines a potentially very aggressive attacker.

$$Spy_1 = in \{\Psi_1\} \cdot out \{\Psi_2\} \cdot in \{\Phi_1\} \cdot out \{\Phi_2\}$$

$$Spy_2 = in \{\Phi_1\} \cdot in \{\Phi_2\} \cdot out \{\Phi_1\} \cdot out \{\Phi_2\}$$

$$Spy_3 = in \{Y\} \cdot Y \cdot in \{\Psi\} \cdot \Psi \cdot out \{\Psi'\} \cdot P_{ub}(\Psi')$$

$$Spy_4 = in \{X\} \cdot \{Priv(X)\} \cdot in \{\Phi'\} \cdot P_{ub}(X) \cdot out \{\Phi'\}$$

$$Spy = \prod_{i \in \{1..4\}} Spy_i$$

The whole system of unlimited dialogues and attacks is obtained by putting in parallel the replicated processes. For NS, it becomes as follows and it can be defined analogously for NSL.

$$INIT = \|_{A,B \in G} Init(A, B)$$

$$RESP = \|_{A \in G} Resp(B)$$

$$NS = INIT \parallel RESP \parallel SPY.$$
can be considered. Also, in order to avoid unreadable transition guards in some figures, we use a shorthand allowing to put complex terms as arc inscriptions (what has no consequences on the behaviour of the net).

3.1 S-net composition operations

The algebra of S-nets has the following three operators:

- The sequence, denoted $N_1 : N_2$: the exit places of $N_1$ are combined with the entry places of $N_2$ (using Cartesian product), all the buffer places having the same label are merged together, the entry places of the sequence are the entry places of $N_1$ and the exit places of the sequence are the exit places of $N_2$.

- The parallel composition, denoted $N_1 || N_2$: $N_1$ and $N_2$ are put side by side and all the buffer places having the same label are merged.

- The replication, denoted $!N$, is defined as an infinite parallel composition, $!N = \|_{i \in \omega} N$.

3.2 Basic S-nets

By analogy with the two basic SPL actions, the in- and out-actions, we introduce two basic S-nets. Each of them has a single transition.

3.2.1 Basic S-net $S_{OUT}(L)$

Let $L = M(\vec{y}, \vec{x}, \vec{\Phi})$ be a generic parameter having the form of an SPL message, where $\vec{y} = \{y_1, y_2, \ldots, y_l\}$ are local name variables, $\vec{x} = \{x_1, x_2, \ldots, x_k\}$ are global name variables and $\vec{\Phi} = \{\Phi_1, \Phi_2, \ldots, \Phi_r\}$ are global message variables.

The basic S-net $S_{OUT}(L)$ is defined as the (unmarked) S-net represented in Figure 1, where the entry and exit places are indicated by incoming/outgoing $\Rightarrow$, the buffer places labelled $y_1, y_2, \ldots, y_l$ and $x_1, x_2, \ldots, x_k$ are of type $Name$, the buffer places labelled $\Phi_1, \Phi_2, \ldots, \Phi_r$ and $\Omega$ are of type $Message$, and the guard of transition $t$ is

$$\gamma(t) = \{ L = M(\vec{y}, \vec{x}, \vec{\Phi}) \land \bigwedge_{1 \leq i \leq l} y_i = a_i \land \bigwedge_{1 \leq k} x_i = b_i \land \bigwedge_{1 \leq r} \Phi_i = c_i \}. $$

We will use the following notational shorthand for the guard assuming that $\gamma(t) = \{ L = M(\vec{a}; \vec{b}, \vec{c}) \}$. Alternatively, we may also replace the arc variable $L$ by the inscription $M(\vec{a}; \vec{b}, \vec{c})$ omitting $\gamma(t)$. This convention will be adopted, in particular, for all examples.

Remark that the arc variables $a_i$ appear free in $\gamma(t)$, so we may assume (in a similar way as in [CW01]) that during the execution, the values (e.g., nonces) which the bindings will assign to, will be chosen freely (in fact, randomly) in the type of adjacent places$^2$.

3.2.2 Basic S-net $S_{IN}(L)$

Let now $L = M(\vec{y}, \vec{x}, \vec{\Phi})$ be a generic parameter having the form of an SPL message, where $\vec{y} = \{y_1, y_2, \ldots, y_l\}$ are local name variables, $\vec{x} = \{x_1, x_2, \ldots, x_k\}$ are local message variables, $\vec{\Psi} = \{\Psi_1, \Psi_2, \ldots, \Psi_m\}$ are global name variables, $\vec{\Phi} = \{\Phi_1, \Phi_2, \ldots, \Phi_r\}$ are global message variables.

The basic S-net $S_{IN}(L)$ is defined as the (unmarked) S-net represented in Figure 2, where the entry and exit places are indicated by incoming/outgoing $\Rightarrow$, the buffer places labelled $y_1, y_2, \ldots, y_l$ and $x_1, x_2, \ldots, x_k$ are of type $Name$, the buffer places labelled $\Phi_1, \Phi_2, \ldots, \Phi_r$ and $\Omega$ are of type $Message$, and the guard of transition $t$ is

$$\gamma(t) = \{ L = M(\vec{a}; \vec{b}, \vec{c}) \}. $$

$^2$Alternatively, we could also consider an explicit mechanism inside the net framework ensuring the freshness of values by considering an additional buffer place, initially marked by a sufficiently large subset of $N$, with input arcs to all transitions OUT, which would deliver different values for each request.
are of type Message, and the guard of transition \( t \) is
\[
\gamma(t) = \{ \begin{array}{l}
L = M(\bar{y}, \bar{\Psi}; \bar{x}, \bar{\Phi}) \land \land_{i \leq l} y_i = a_i \\
\land_{i \leq m} \Psi_i = d_i \land \land_{i \leq k} x_i = b_i \land \land_{i \leq r} \Phi_i = e_i
\end{array} \}.
\]
Its abbreviated version is \( \gamma(t) = \{ L = M(\bar{a}, \bar{d}; \bar{b}, \bar{c}) \} \).

4 Semantics of SPL processes

The S-net semantics of SPL processes is defined through the semantic function \( \text{Snet} \). We proceed compositionally following the structure of processes \( p \) assuming that all SPL processes are well-formed.

4.1 Semantics of the out action \( \text{out new } \bar{y} M \)

Let \( \text{out new } \bar{y} M \) be an SPL out-action. By definition, the variables in \( \bar{y} \) are local in \( M \), while all the other variables appearing in \( M \), \( \bar{x} \) for names and \( \bar{\Phi} \) for messages, are global. So, let us assume \( L = M(\bar{y}; \bar{x}, \bar{\Phi}) \) and define
\[
\text{Snet}(\text{out new } \bar{y} M) = \text{SOUT}(M(\bar{y}; \bar{x}, \bar{\Phi})).
\]

Example: We illustrate this construction for the elementary SPL action \( \text{out new } \{ \bar{y}, A, z \} \). Thus, the parameter \( L \) is instantiated with \( M(\bar{y}; z) \) (with \( \bar{\Phi} = \emptyset \)). In Figure 3, following a convention, the variable \( L \) is replaced by the message \( \{ a, A, b \} \) allowing to omit the guard. This net may be executed if there is a token \( \bullet \) in the entry place, and if the buffer place labelled \( z \) is marked (i.e., contains a value (nonce)). At the firing of \( t \) under \( \sigma \) a new nonce is generated and put in place \( y \), the token existing in place \( z \) is read (but not removed) and the whole message \( \{ \sigma(a), A, \sigma(b) \} \) is put in the message buffer \( \Omega \).

Figure 3. The S-net of \( \text{out new } \{ \bar{y}, A, z \} \).

4.2 Semantics of the in-action \( \text{in pat } \bar{y} \bar{\Psi} M \)

Let \( \text{in pat } \bar{y} \bar{\Psi} M \) be an SPL in-action. By definition, in that context, the variables \( \bar{y} \) and \( \bar{\Psi} \) are local in \( M \), while all the other variables appearing in \( M \), \( \bar{x} \) for names and \( \bar{\Phi} \) for messages, are global.

Before defining the corresponding S-net semantics, let us remark that in some cases, we need to implement explicitly in the basic net the public key device. For instance, if the in-action is a part of the process defining an agent \( A \), we have to assume that \( \text{Priv}(A) \) is known and take this information into account. In the net framework this means that we should have an additional condition in \( \gamma(t) \): “if the read message is encrypted by a public key then the key has to be \( \text{Pub}(A) \)". The whole guard of \( t \) (in its non abbreviated version), now called \( \gamma_A(t) \), then becomes
\[
\gamma_A(t) = \{ \begin{array}{l}
L = M(\bar{y}, \bar{\Psi}; \bar{x}, \bar{\Phi}) \land \land_{i \leq l} y_i = a_i \\
\land_{i \leq m} \Psi_i = d_i \land \land_{i \leq k} x_i = b_i \land \land_{i \leq r} \Phi_i = e_i
\end{array} \}.
\]

Then, assuming \( L = M(\bar{y}, \bar{\Psi}; \bar{x}, \bar{\Phi}) \), we define
\[
\text{Snet}(\text{in pat } \bar{y} \bar{\Psi} M) = \text{SIN}(M(\bar{y}, \bar{\Psi}; \bar{x}, \bar{\Phi})) \text{ with } \gamma_A(t).
\]

In figures, for a better readability, we take the same shorthand of the condition as that in Figure 2 and indicate, by indexing the transition name with \( A \), \( t \text{IN}_A \), the existence of the additional condition concerning \( A \)’s key.

We do not use the indexed form if the message is not encrypted or if there is no owner agent known, like for \( \text{Spy} \) processes. In such cases we define
\[
\text{Snet}(\text{in pat } \bar{y} \bar{\Psi} M) = \text{SIN}(M(\bar{y}, \bar{\Psi}; \bar{x}, \bar{\Phi})) \text{.}
\]

Examples: We illustrate this construction for the non encrypted in-action \( \text{in pat } \{ \bar{y}, \bar{\Psi} \} \{ \bar{y}, z, B, \bar{\Psi} \} \).

The message \( M \) has the local variables \( y \) and \( \Psi \), a global variable \( z \) and no global message variable. Thus, the parameter \( L \) has to be instantiated by \( M(\bar{y}, \bar{\Psi}; z) \). In Figure 4, the variable \( L \) is replaced by the incoming message \( \{ a, d, B, b \} \) and thus, \( \gamma(t) = \emptyset \) and can be omitted. At the firing of \( t \), a message from \( \Omega \) with the same pattern is read such that the existing value of \( z \) in place \( z \) is checked (through the read-arc) if it is equal to the second element of the message, and the values of the variables \( y \) and \( \Psi \) are decoded from the message and put in places \( y \) and \( \Psi \).

Figure 4. The S-net of \( \text{in pat } \{ \bar{y}, \bar{\Psi} \} \{ \bar{y}, z, B, \bar{\Psi} \} \).

The same in-action but with encrypted message \( \text{in pat } \{ \bar{y}, \bar{\Psi} \} \{ \bar{y}, z, B, \bar{\Psi} \}_{\text{Pub}(A)} \) in the specification of the agent \( A \) is modelled by the S-net represented in Figure 5.
5.1 Modelling of NS with S-nets

As presented in section 2, the SPL processes describing the two NS or NSL agents, \( A \) and \( B \), are both expressed by sequential processes composed of three actions. In the simple case where only \( A \) and \( B \) converse, they are just put in parallel. These protocols are specified by closed processes in SPL, moreover we always assume that they are well formed. Thus the only merged places by parallel composition will be the \( \Omega \) places of the two sub-nets.

Figure 6 shows the result of the translation of the simple NS protocol into an S-net. The S-net for \( \text{Init}(A, B) \) is represented on the left hand side, while \( \text{Resp}(B) \) is on the right hand side, both being connected to the message buffer place \( \Omega \), represented in the middle.

5.2 Non-authentication of NS

To show that the NS protocol is prone to an attack, we have to proof that the \( \text{Spy} \) processes can perform successfully an attacker, e.g., the third man \( C \) in the middle, described as follows: \( A \) thinks that \( C \) is honest and performs with him the simple NS protocol. \( C \) pretends to be \( A \) with respect to \( B \), \( B \) is cheated and responds to \( A \). Thus, \( C \) succeeds to discover the nonce of \( B \) and will be able in the sequel to eavesdrop and falsify the communications between \( B \) and \( A \). The informal protocol is as follows:

1. \( A \rightarrow C : \{m, A\}^{\text{Pub}(C)} \quad C \rightarrow B : \{m, A\}^{\text{Pub}(B)} \)
2. \( B \rightarrow A : \{m, n\}^{\text{Pub}(A)} \quad C \rightarrow B : \{n\}^{\text{Pub}(B)} \)

In fact, \( C \) is not a honest agent performing his responder part, but consents to a certain combination of \( \text{Spy} \) processes to misuse his keys in order to play the role of above attacker \( C \). To allow the use of automated tools, a finite version of NS should be considered. It has to be chosen large enough in order to be able to show that a certain marking expressing a violation is reachable. The firing sequence we look for to reach such a marking will be necessarily finite and thus, involves only a finite part of the whole S-net for NS. To establish the proof we only need to consider this concerned part of the net.

5.2.1 The concerned part of the protocol and the corresponding S-net

We found that in a configuration of just one initiator \( \text{Init}(A, C) \) and one responder \( \text{Resp}(B) \), two \( \text{Spy}_3 \)s and two \( \text{Spy}_4 \)s processes are sufficient for the role of attacker. This quite small part is called \( \text{NS}_f \) and is formally defined by the following SPL process:

\[
\text{NS}_f = \text{Init}(A, C) \parallel \text{Resp}(B) \parallel \text{Spy}_3 \parallel \text{Spy}_3' \parallel \text{Spy}_4 \parallel \text{Spy}_4',
\]

where \( \text{Spy}_3 \) and \( \text{Spy}_4 \) are just renamed versions of \( \text{Spy}_3 \) and \( \text{Spy}_4 \), respectively. Both processes \( \text{Init}(A, C) \) et \( \text{Resp}(B) \) need to perform completely their protocol. In net terms, this is expressed by the requirement that the sub-nets for these two processes reach their final marking, i.e., have a token in their exit places. We would have authentication if the last message received by \( B \) confirming it its nonce \( n \) has been well received by \( A \), is proved to come necessarily from process \( \text{Init}(A, B) \). The authentication is violated, if there are other possibilities for the origin of the message, and if \( n \) has circulated in clear. As the global buffer place \( \Omega \) stocks all messages which have circulated in the network, we should find in \( \Omega \) several messages with \( n \) as \( \{n\}^{\text{Pub}(X)}, \{n\}^{\text{Pub}(B)} \) for some \( X \) being an attacker, i.e., whose name and private key is also leaked and thus, present as tokens in the place \( \Omega \). A marking with such tokens will be called violating.
5.2.2 Notations concerning Snet(NS_f)

Figure 7 shows the result of the translation of NS_f into Snet(NS_f). We adopted some particular rules of notation when designing this net and naming its objects, making clearer later on the argumentation and proofs on the net: Each of the six sub-nets received a different name: A for Snet(Init(A, C)), B for Snet(Resp(B)), W and X for Snet(Spy_3) and Snet(Spy_4), finally Y and Z for Snet(Spy_5) and Snet(Spy_6). For a better readability we present the involved nets separately, even if they share the buffer places in each net, which are enumerated p^1, p^2, (p^3), we can and do omit their labels (variable names) in the figure here.

Places: The places of this net are all enumerated in the same way and systematically indexed by the name of the concerned sub-net:

- For each sub-net having four control places in sequence, they are enumerated from top to down by 1 (entry place) to 4 (exit place), e.g., from 1_A to 4_A in net A.
- There are two or three buffer places in each net, which are enumerated p^1, p^2, (p^3), we can and do omit their labels (variable names) in the figure here.

Transitions and evolutions: In each sub-net the transitions are enumerated from top to down by t^1, t^2, t^3, indexed by the name of the concerned sub-net. As introduced in section 3.2 they also wear IN resp. OUT to indicate their origin. If, at marking M, the firing of some transition t_K^σ of the sub-net K under the binding σ yields the new marking M', we write M t_K^σ ⇒ M'. We extend this notation to alternating firing sequences:

\[ M t_1^σ t_2^σ t_3^σ t_4^σ \ldots \]

The initial marking M^0: All entry places are initially marked and the place \( Ω \) contains the message tokens \( \{A\}, \{B\}, \{C\}, \{Priv(C)\} \). This marking expresses that the public keys of the three agents supposed honest are known to everybody and that C is in reality immoral and tolerates that its private key \( Priv(C) \) could be used (in fact, misused) by the \( Spy \) processes.

A violating final marking: Such a marking, as explained above, should contain in the place \( Ω \) at least the following tokens \( \{C\}, \{Priv(C)\}, \{n\}, \{n\}_Pub(C), \{n\}_Pub(B), \{n\}_Pub(B) \), as only C can be misused for the attacker role.

5.2.3 The violating alternating firing sequence

We exhibit in the appendix the found evolution of this S-net. It leads from M^0 to a violating final marking, where the exit places of the sub-nets A et B are marked and where the place \( Ω \) contains the tokens \( \{C\}, \{Priv(C)\}, \{n\}, \{n\}_Pub(C), \{n\}_Pub(B) \), we were looking for. Thus, we have shown that a violating final marking is reachable, which proves the violation of authentication of NS.

Our approach may be also useful in interpreting the found (eighteen transitions long) firing sequence. One can read off, for instance, that all six sub-nets are completely executed. It also allows to observe, how the Spies collaborate to imitate "the man in the middle" (we refer to the informal protocol given in section 5.2): the C-part in (1) is played by Spy_3 (in net Y); it picks the private key of C, and uses it to decode the message from A. Then, Spy_5 (in net W) forwards it to B. In part (2): B responds to A, that receives the message. In part (3): Spy_4 (in net Z) decodes now the message from A to C and Spy_6 (in net X) forwards it to B.
(1) Le S-net A for Init(A, C)

(2) Le S-net B for Resp(B)

(3) The S-net W for Spy₃ (and its copy X)

(4) The S-net Y for Spy₄ (and its copy Z)

Figure 7. Some S-nets involved in Snet(NS).
This corresponds to a sequential execution $\text{Spy}_4 - \text{Spy}_3 - \text{Spy}_4' - \text{Spy}_3'$ of the parallel composition $\text{SPY}$.

5.3 Property of authentication of NSL

To prove the property of authentication of the complete NSL protocol it is sufficient to show that no execution of the S-net associated to it, $\text{Snet}(\text{NSL})$, leads to a violating marking. As the protocol uses an infinite construction, due to replication, the S-net is also infinite, which limits the use of automated tools. Thus, we can still show the property by contradiction, as for instance in [CW01]: if we assume to have an evolution of the net which reaches a forbidden marking (violating authentication) while $\text{Init}'$ and $\text{Resp}'$ have reached normal final marking, we are able to show that this is impossible as the net is already in deadlock earlier: it could never fire the $\text{IN}$-transition of the sub-net for agent $A$. We can conclude, that no violating marking is reachable and so that NSL protocol possesses the property of authentication.

6 Conclusion

In this work we introduced a particular, small class of composable high-level Petri nets, dedicated to security protocols, the S-nets. We defined how to associate an S-net to a formally specified protocol in the language SPL, such that the different behaviours of the S-net correspond exactly to the different executions or runs of the protocol. In other approaches, as in [NT93, CW01], one net corresponds only to one particular run of the protocol, thus an infinity of nets have to be built for the whole semantics.

We illustrated our method using the Needham-Schröder protocol, which served us as a running example, including how to model an attack by a third man in the middle. We showed that our method may be interesting in the proof, that such an attack can be successful, by exhibiting a firing sequence of the S-net, which established the violation of the authentication property.

As expected, and as other automata-based models, this approach has only a little importance with respect to automated techniques concerning proofs of correctness of infinitary security protocols. Its real strength is certainly in detecting errors in protocols, when designing new ones or checking complicated applied ones, as those presented in [S96].

References


Appendix (will be removed from the paper, if accepted)

A  The violating alternating firing sequence of $S_{\text{net}}(NS_f)$

In order to be able to read and interpret the alternating firing sequence below, we adopt some particular rules allowing writing down complex markings and their changes. The markings $M$ of the sub-nets of the S-net $S_{\text{net}}(NS_f)$, shown in section 5.2, are also systematically indexed by the name of the sub-net, and the marking of the whole S-net can be written as concatenation of the markings of its sub-nets $M = M_A M_B M_W M_X M_Y M_Z M_\Omega$:

- The marking of place $\Omega$ will only increase because $\Omega$ has no outgoing arc, thus successor marking $M'_\Omega$ can be written by adding the new tokens produced by a firing: $M'_\Omega = M_\Omega + L$ where $L$ stands for the new message tokens.

- The marking of the S-net $A$ is given as a list:
  
  $M_A = [\text{ordered list of its non empty buffer places with their token; number of the unique non empty control place }]_A,$
  
  the index $A$ being shifted outside the brackets. E.g., $M_A = [p_1 : \text{val}_1, p_2 : \text{val}_2; 2]_A$ says that place $p_1$ contains one token $\text{val}_1$, place $p_2$ contains one token $\text{val}_2$ and place $2_A$ contains the marking $M'.$

- If the firing of $t$ at marking $M$ leads to a new marking $M'$ we indicate in $M'$ explicitly the changed sub-markings, but write $M_X$ if $M_X = M_X$. E.g., the full marking: $M' = [M_A[p_1 : \text{val}; 2]_B M_W M_X M_Y M_Z M_\Omega + \{ \Psi \}$ has the following meaning: The marking of S-net $B$ has changed and is given explicitly, those of S-nets $A, W, X, Y, Z$ are unchanged, the marking $M'$ of $\Omega$ is that of $M$ increased by message $\{ \Psi \}$.

As fixed in section 5.2, the initial marking of $\Omega$ is $M_\Omega = \{ \{ A \}, \{ B \}, \{ C \}, \{ Priv(C) \} \}$ and the entry places $[1]_K$ of all six sub-nets $K$ contain one token each. We are exhibiting the found evolution of the net leading to a violating final marking and adding as comment when the $Spy$ processes are interfering in the normal behaviour of NS agents $A$ and $B$.

Thus we start with the initial marking

$$M^0 = [1]_A [1]_B [1]_W [1]_X [1]_Y [1]_Z M^0_\Omega$$

with the binding

$$\tau^1_{A,\sigma_3}: M^1 = [p_1 : m; 2]_A M_B M_W M_X M_Y M_Z M_\Omega + \{ m, A \} P_{\text{ub}}(C)$$

Comment: $Spy_4$ will start

$$\tau^2_{A,\sigma_2}: M^2 = M_A M_B M_W M_X [p_1 : C; 2]_Y M_Z M_\Omega$$

$$\sigma_2(d) = C$$

$$\tau^3_{A,\sigma_3}: M^3 = M_A M_B M_W M_X [p_1 : C, p_2 : \{ m, A \}; 3]_Y M_Z M_\Omega$$

$$\sigma_3(d) = C, \sigma_3(d') = \{ m, A \}$$

$$\tau^4_{A,\sigma_3}: M^4 = M_A M_B M_W M_X [p_1 : C, p_2 : \{ m, A \}; 4]_Y M_Z M_\Omega + \{ m, A \}$$

$$\sigma_4(d') = \{ m, A \}$$

Comment: $Spy_4$ did his work and $Spy_3$ will start

$$\tau^5_{W,\sigma_5}: M^5 = M_A M_B [p_1 : \{ B \}; 2]_W M_X M_Y M_Z M_\Omega$$

$$\sigma_5(c) = B$$

$$\tau^6_{W,\sigma_6}: M^6 = M_A M_B [p_1 : \{ B \}, p_2 : \{ m, A \}; 3]_W M_X M_Y M_Z M_\Omega$$

$$\sigma_6(c') = \{ m, A \}$$

$$\tau^7_{W,\sigma_7}: M^7 = M_A M_B [p_1 : \{ B \}, p_2 : \{ m, A \}; 4]_W M_X M_Y M_Z M_\Omega + \{ m, A \} P_{\text{ub}}(B)$$

$$\sigma_7(c') = \{ m, A \}, \sigma_7(c) = B$$

Comment: $Spy_3$ did his work

$$\tau^8_{A,\sigma_8}: M^8 = M_A [p_1 : m, p_2 : A; 2]_B M_W M_X M_Y M_Z M_\Omega$$

$$\sigma_8(b) = m, \sigma_8(b') = A$$

$$\tau^9_{A,\sigma_9}: M^9 = M_A [p_1 : m, p_2 : A, p_3 : n; 3]_B M_W M_X M_Y M_Z M_\Omega + \{ m, n \} P_{\text{ub}}(A)$$

$$\sigma_9(b) = m, \sigma_9(b') = A, \sigma_9(b'') = n$$

$$\tau_{A,\sigma_{10}}: M^{10} = [p_1 : m, p_2 : n; 3]_A M_B M_W M_X M_Y M_Z M_\Omega$$

$$\sigma_{10}(a) = m, \sigma_{10}(a') = n$$

$$\tau_{A,\sigma_{11}}: M^{11} = [p_1 : m, p_2 : n; 4]_A M_B M_W M_X M_Y M_Z M_\Omega + \{ n \} P_{\text{ub}}(C)$$

$$\sigma_{11}(a') = n$$
Comment: Spy′ will start

\[
\begin{align*}
\text{\textcolor{red}{\sigma_{12}(d) = C}} & \\
\text{\textcolor{red}{\sigma_{13}(d) = C, \sigma_{13}(d') = n}} & \\
\text{\textcolor{red}{\sigma_{14}(d') = n}} &
\end{align*}
\]

Comment: Spy′ did his work and Spy′ will start

\[
\begin{align*}
\text{\textcolor{red}{\sigma_{15}(c) = B}} & \\
\text{\textcolor{red}{\sigma_{16}(c') = n}} & \\
\text{\textcolor{red}{\sigma_{17}(c) = B, \sigma_{17}(c') = n}} &
\end{align*}
\]

Comment: Spy′ did his work

\[
\begin{align*}
\text{\textcolor{red}{\sigma_{18}(b'') = n}} &
\end{align*}
\]

In the reached marking \(M^{18}\) the exit places of the sub-nets A and B, 4A and 4B, are marked, thus the protocol is finished for agents A and B, which was one of the requirements for the final marking. Now, let us check the marking of place Ω:

\[
M^{18}_{\Omega} = (\{A\}, \{B\}, \{C\}, \{Priv(C)\}, \{m, A\}_{Pub(C)}, \{m, A\}_{Pub(A)}, \{m, n\}_{Pub(A)}, \{n\}, \{n\}_{Pub(B)}, \{n\}_{Pub(B)})
\]

The messages \{C\}, \{Priv(C)\}, \{n\}, \{n\}_{Pub(C)}, \{n\}_{Pub(B)}, we are looking for, are in \(M^{18}_{\Omega}\). Thus, we have shown that a violating final marking, here \(M^{18}\), is reachable, which proves the violation of authentication of NS.