# Primitives for Authentication in Process Algebras *

Chiara Bodei, Pierpaolo Degano
Dipartimento di Informatica, Università di Pisa
Corso Italia, 40, I-56125 Pisa, Italy
{chiara,degano}@di.unipi.it

Riccardo Focardi
Dipartimento di Informatica, Università Ca' Foscari di Venezia
Via Torino 155, I-30173 Venezia, Italy
focardi@dsi.unive.it

Corrado Priami
Dipartimento Scientifico Tecnologico, Università di Verona
Ca' Vignal 2, Strada Le Grazie, 15, I-37134 Verona, Italy
priami@sci.univr.it

**Abstract**

We extend the $\pi$-calculus and the spi-calculus with two primitives that guarantee authentication. They enable us to abstract from various implementations/specifications of authentication, and to obtain idealized protocols which are "secure by construction". The main underlying idea, originally proposed in [14] for entity authentication, is to use the *locations* of processes in order to check who is sending a message (authentication of a party) and who originated a message (message authentication). The theory of local names, developed in [6] for the $\pi$-calculus, gives us almost for free both the partner authentication and the message authentication primitives.

**Keywords** Security, Authentication, Secrecy, Operational Semantics, Proved Transition Systems, Distributed Process Algebras.

## 1 Introduction

Authentication is one of the main issues in security and it can have different purposes depending on the specific application considered. For example, *entity authentication* is related to the verification of an entity's claimed identity [18], while *message authentication* should make it possible for the receiver of a message to ascertain its origin [30]. In recent years there have been some formalizations of these different aspects of authentication (see, e.g., [3, 7, 13, 16, 17, 22, 29]). These formalizations are crucial for proofs of authentication properties, that sometimes have been automatized (see, e.g. [12, 15, 20, 21, 25]). A typical approach presented in the literature is the following. First, a protocol is specified in a certain formal model. Then

*An earlier version of Sections 7-9 appeared in [8]. This work has been partially supported by MURST Progetto TOSCA and Progetto "Certificazione automatica di programmi mediante interpretazione astratta".

the protocol is shown to enjoy the desired properties, regardless of its operating environment, that can be unreliable, and can even harbour a hostile intruder.

We use here basic calculi for modelling concurrent and mobile agents and we give then certain kinds of semantics, offering built-in mechanisms that guarantee authentication. This is the main contribution of our paper. Our mechanisms enable us to abstract from the various implementations/specifications of authentication, and to obtain idealized protocols which are "secure by construction". Our protocols, or rather their specifications can then be seen as a reference for proving the correctness of "real protocols".

The essence of concurrent and mobile computation can be studied in a pure form using the $\pi$-calculus [24], a foundational calculus based on the notion of naming. Systems are specified as expressions called *processes*. These are obtained by combining, via a few operators (parallel composition, nondeterministic choice, declarations), the basic actions of sending and of receiving names between processes along channels. Names represent values, or messages, and also channels. Since processes exchange names in communications, the interconnection structure of a network can vary dynamically. Recently, Abadi and Gordon [3] defined the spi-calculus by enriching the $\pi$-calculus with primitives for encryption and decryption. The resulting calculus is particularly suited to security issues, among which authentication.

In [6] the $\pi$-calculus has been equipped with a structural operational semantics which endows each sequential process $P$ in the whole system with its own *local* environment, i.e., $P$ has its *local* space of names and its *local* name manager that generates a fresh name, whenever necessary. The basic ingredient of this proposal is the notion of *relative address* of a process $P$ with respect to another process $Q$: it represents the path between $P$ and $Q$ in (an abstract view of) the network (as defined by the syntax of the calculus). Note that relative addresses are *not* available to the users of the $\pi$-calculus: they are used by the abstract machine of the calculus only, defined by its semantics.

We propose here to use the ideas underlying this proposal to study authentication, both in the $\pi$-calculus and in the spi-calculus. As a matter of fact, this kind of semantics provides us with two built-in authentication primitives. The key point is that $P$ can use its address relative to $Q$ to uniquely reach (the subterm of the whole system representing) $Q$ itself. Consequently, relative addresses may be used both to authenticate the partners of a communication and to authenticate the origin of a message. For the sake of presentation, we will first introduce our primitive for partner authentication in the $\pi$-calculus, and the one of message authentication in the spi-calculus. We can easily combine them, e.g. by introducing the first mechanism in the spi-calculus so that both kinds of authentication can be enforced.

**Partner authentication** A variation of the semantics defined in [6] gives us a run-time mechanism that guarantees each principal to engage an entire run session with the same partners, playing the same roles. Essentially, we bind sensitive input and output communications to a relative address, i.e. a process $P$ can accept communications on a certain channel, say $c$, only if the relative address of its partner is equal to an a-priori fixed address $loc$.

In order to achieve this, we index channels with relative addresses, so obtaining input actions on the form $c_{loc}(x)$ or output actions of the form $\overline{d}_{loc'}\langle M\rangle$. While sending a message, our semantics will check if the address of the sender with respect to the receiver is indeed $loc$. In particular, assume that $P$ is explicitly waiting a message from a process reachable following $loc$, i.e. $P$ performs the input action $c_{loc}(x)$. Then, no possibly hostile process $E$ having a relative address with respect to $P$ different from $loc$, can successfully communicate with $P$ on $c$. Moreover, $E$ cannot sniff any message $M$ sent by $P$ through the output action $\overline{d}_{loc'}\langle M\rangle$, if the address of $E$ relative to $P$ is different from $loc'$. These "located" I/O primitives enable processes to have some control on their partners. As an example we can define, in the $\pi$-calculus syntax, a protocol that guarantees partner authentication by construction by putting in parallel the following processes:

$$\begin{array}{rcl} P & = & \overline{c}\langle M\rangle.P' \\ Q & = & c_{loc_P}(x).Q' \end{array}$$

where $loc_P$ represents the address of $P$ relative to $Q$, $\overline{c}\langle M\rangle$ stands for sending $M$ along $c$ to every possible process (the channel $c$ is not indexed by any relative address, formally by the empty one), and $c_{loc_P}(x)$ is an input action located at $loc_P$. The input can only match an output $\overline{c}\langle M\rangle$ executed by the process reachable from $Q$ through the relative address $loc_P$. The resulting communication has the effect of binding $x$ to $M$ within the residual of $Q$, yielding $Q'[M/x]$ (i.e. $Q'$ where $M$ replaces $x$).

If we consider $P$, $Q$ and also an intruder $E$ in parallel, $(P \mid Q) \mid E$, the effect is to guarantee to $Q$ that the communication over $c$ can only be performed with $P$. Thus, $Q$ is assured that message $M$ has been indeed received by $P$. Note that there is no need for $c$ to be a channel private to $P$ and $Q$.

Although in this paper we focus on authentication primitives, it is interesting to note that located outputs also guarantee a form of secrecy. As an example consider the following protocol where now $P$ uses a located output:

$$\begin{array}{rcl} P & = & \overline{c}_{loc_Q}\langle M\rangle.P' \\ Q & = & c_{loc_P}(x).Q' \end{array}$$

where $loc_Q$ is the address of $Q$ relative to $P$. Consider again $(P \mid Q) \mid E$. Now, $P$ is also guaranteed that the communication over $c$ will be only performed with $Q$, i.e., $E$ cannot intercept $M$ which will thus remain secret. Again, the channel $c$ needs not to be private to $P$ and $Q$. So, we separately model authentication and secrecy over public channels: our mechanism is thus more concrete than the use of a private channel for communication.

In every protocol, legitimate processes may play a few different roles, such as sender, server, receiver, etc. Usually, processes recognize the roles their partners are playing, but seldom they know which are the partners' relative addresses. So, we shall also index a channel with a variable $\lambda$, to be instantiated by a relative address, only. Whenever a process $P$, playing for instance the role of sender or initiator, has to communicate for the first time with another process $S$ in the role, e.g. of server, it uses a channel $c_\lambda$. Our semantics rules will take care of instantiating $\lambda$ with the address of $P$ relative to $S$ during the communication. Roughly, this implements a sort of anonymous communication. Note that the process $S$ could also be a hostile process pretending to be the server. However, from that point on, $P$ and $S$ will keep communicating in the same roles for the entire session, using their relative addresses.

We have sketched how our mechanism guarantees that each principal communicates with the same partners, in the same role, for the entire session. Through it, we also circumvent some problems arising from mixing up sessions, in particular, those due to replay attacks. Usually, protocols use challenge-response mechanisms, based e.g. on nonces (typically numbers used once), whose freshness protects from replay attacks. In our framework, freshness is no longer needed to distinguish the communications of one session from the communications of another.

**Message authentication** The semantics in [6] and its extension to the spi-calculus studied in Section 7 (see also [5]) directly allow to define another authentication mechanism, providing us with a built-in primitive that enables the receiver of a message to ascertain its origin, i.e. the process that created it. In fact, the address of a message $M$ relative to a process $P$, uniquely identifies the originator of $M$, even after the message has been maliciously intercepted and forwarded. Indeed, we guarantee the integrity of the message

$M$: its receiver gets it as the originator of $M$ made it. If $M$ is a compound message, the receiver can additionally ascertain the originators of the components of $M$. We write the primitive for authentication as $[M \overset{@}{=} P]Q$, where $M$ is a message and $P$ is a process.[1] Intuitively, the execution of the process $Q$ starts only when the check $[M \overset{@}{=} P]$ succeeds, and this happens if and only if the relative addresses of $M$ and of $P$ with respect to $Q$ coincide: in other words, $Q$ is the generator of $M$. Note again that this check is done by the interpreter of the calculus, i.e. the semantic rules, *not* by the user.

A communication protocol that guarantees message authentication by construction is now easy to define, by using the plain handshaking communication of the $\pi$-calculus and the spi-calculus, and the primitive sketched above. Suppose that a process $P$ sends a message $M$ (for simplicity we consider below a name) to $Q$ along a public channel $c$. In the spi-calculus syntax they have the following form:

$$P = (\nu M)\overline{c}\langle M\rangle.P' \quad \text{and} \quad Q = c(x).[x \overset{@}{=} P]Q',$$

where the operator $(\nu M)$ declares $M$ to be a fresh name, different from all the others in the whole system. If we put $P$, $Q$ and also an intruder $E$ in parallel, $(P \mid Q) \mid E$, the effect is to guarantee that the residual of $Q$ is indeed $Q'[M/x]$. Note that here $Q$ is not guaranteed to receive the message *directly* from $P$, as for partner authentication. As a matter of fact, the intruder might as well have intercepted the message $M$ originated by $P$ and forwarded it to $Q$. This is legal as we are only checking that $M$ has been originated by $P$. As we will see, $E$ cannot modify any of the parts of $M$ without changing the relative address of $M$ itself, because relative addresses are manipulated by the semantics only. Also in this case, there is no need for $c$ to be a channel private to $P$ and $Q$.

Our solutions assume that the implementation of the communication primitives has a reliable mechanism to control and manage relative addresses. In some real cases this is possible, e.g., if the network management system filters every access of a user to the network as it happens in a LAN or in a virtual private network. This may not be the case in many other situations. However, relative addresses can be built by storing the actual address of processes in selected, secure parts of message headers (cf. IPsec [31]). Yet, our solutions may help checking the correctness of different implementations, e.g. those based on cryptography, as briefly discussed in the conclusion.

**Contents of this paper** In Section 2 we survey the $\pi$-calculus; in Section 3, we recall relative addresses and in Section 4 the proved version of the $\pi$-calculus from [6]. Section 5 is devoted to partner authentication. In Section 6 we survey the spi-calculus and in Section 7 we enrich it with the relative address mechanism. Sections 8 and 9 are about the message authentication primitive.

## 2 The $\pi$-calculus

In this section we briefly recall the monadic $\pi$-calculus [24], a model of concurrent communicating processes based on the notion of *naming*. Our presentation slightly differs from the usual ones and it will make it easier to introduce later on the spi-calculus The main difference from standard presentation relies in the introduction of the new syntactic category of terms, where names and variables are distinguished.

**Definition 2.1 (syntax)** Terms *(denoted by $M, N, \ldots \in \mathcal{T}$) and* processes *(denoted by $P, Q, R, \ldots \in \mathcal{P}$) are built according to the syntax*

[1] Actually, we have $[M \overset{@}{=} N]Q$, where $N$ is a message from $P$; see the formal development for details.

$M ::=$ *terms*
$a, b, m, n, \ldots \in \mathcal{N}$ *names*
$x, y, \ldots \in \mathcal{V}$ *variables*

$P ::=$ *processes*
$\mathbf{0}$ *nil*
$\pi.P$ *prefix*
$P + P$ *summation*
$P|P$ *parallel composition*
$(\nu m)P$ *restriction*
$[M = N]P$ *matching*
$!P$ *replication*

*where $\pi$ may either be $M(x)$ for* input *or $\overline{M}\langle N\rangle$ for* output.
*Hereafter, the trailing* **0** *will be omitted. We often write $\tilde{\cdot}$ to denote tuples of objects, for instance $\tilde{m}$ for the vector $(m_1, \ldots, m_r)$; actually we feel free to consider and to operate on $\tilde{m}$ as if it were a set. Notations are extended componentwise, e.g. $(\nu\tilde{n})$ stands for $(\nu n_1), \ldots, (\nu n_r)$; we assume $(\nu m)(\nu\tilde{n}) = (\nu m, \tilde{n})$; finally, $(\nu)$ means that there are no restricted names.*

Intuitively, **0** represents the null process which can do nothing. The prefix $\pi$ is the first atomic action that the process $\pi.P$ can perform. After the execution of $\pi$ the process $\pi.P$ behaves like $P$. The input prefix $M(x)$ binds the name $x$ in the prefixed process as follows: when a name $N$ is received along the link named $M$, all the (free) occurrences of $x$ in the prefixed process $P$ are substituted with $M$. The output prefix $\overline{M}\langle N\rangle$ does not bind the name $N$ which is sent along $M$. Summation denotes nondeterministic choice. The process $P_1 + P_2$ behaves like either $P_1$ or $P_2$. The operator $|$ describes parallel composition of processes. The components of $P_1|P_2$ may act independently; also, an output action of $P_1$ (resp. $P_2$) at any output port $\overline{M}$ may synchronize with an input action of $P_2$ (resp. $P_1$) at $M$. The value sent by $P_1$ replaces the relevant occurrences of the placeholder $x$ in $P_2$. The operator $(\nu m)$ acts as a static declaration (i.e. a binder for) the name $m$ in the process $P$ that it prefixes. In other words, $m$ is a unique name in $P$ which is different from all the external names. The agent $(\nu m)P$ behaves as $P$ except that actions at ports $\overline{m}$ and $m$ are prohibited. However communications along link $m$ of components *within* $P$ are allowed. Matching $[M = N]P$ is an `if-then` operator: process $P$ is activated only if $M = N$. Finally, the process $!P$ behaves as infinitely many copies of $P$ running in parallel.

We write *fn*$(M)$ and *fn*$(P)$ for the sets of names free in term $M$ and process $P$, respectively, and *fv*$(M)$ and *fv*$(P)$ for the sets of variables free in term $M$ and process $P$, respectively. A *closed* term or process is a term or process without free variables.

### 2.1 Semantics.

The semantics for the $\pi$-calculus we consider here is a late semantics, based on a reduction relation and on a commitment relation. Some structural congruence rules are also needed. The commitment relation depends on the *abstraction* and *concretion* constructs:

- An abstraction has the form $(x)P$, where $(x)$ binds $x$ in $P$.
- A concretion has the form $(\nu\tilde{m})\langle M\rangle P$, where $M$ is a term, $P$ is a process and the names in $\tilde{m}$ are bound by $(\nu\tilde{m})$ in $M$ and $P$.

An *agent* $A$ or $B$ is an abstraction, a concretion or a process.

If $F$ is the abstraction $(x)P$ and $C$ the concretion $(\nu\tilde{m})\langle M\rangle Q$ and $\{\tilde{m}\}\cap fn(P) = \emptyset$, then the interactions $F@C$ and $C@F$ are:

$$F@C = (\nu\tilde{m})(P[M/x] \mid Q)$$
$$C@F = (\nu\tilde{m})(Q \mid P[M/x])$$

**Congruence.** The *structural congruence* $\equiv$ on processes is defined in the standard way, except for the treatment of parallel composition that is assumed to be neither commutative nor associative. It is then defined to be the least congruence satisfying:

- if $P$ and $Q$ are $\alpha$-equivalent then $P \equiv Q$;
- $(\mathcal{P}/_{\equiv}, +, \mathbf{0})$ is a commutative monoid;
- $(\nu m)(P_1|P_2) \equiv (\nu m)P_1|P_2$ and $(\nu m)(P_2|P_1) \equiv P_2|(\nu m)P_1$ if $m \notin fn(P_2)$, and $(\nu m)P \equiv P$ if $m \notin fn(P)$.
- $(\nu m)(x)P = (x)(\nu m)P$
- $R \mid (x)P = (x)(R \mid P)$ and $(x)P \mid R = (x)(P \mid R)$, if $x \notin fv(R)$
- $R \mid (\nu\tilde{m})\langle M\rangle Q = (\nu\tilde{m})\langle M\rangle(R \mid Q)$, and $(\nu\tilde{m})\langle M\rangle Q \mid R = (\nu\tilde{m})\langle M\rangle(Q \mid R)$, if $\{\tilde{m}\} \cap fn(R) = \emptyset$,

In the following, we will never distinguish congruent terms.

**Reduction relation.** The *reduction relation* $>$ is the least relation on closed processes that is transitive and closed under all contexts, and that satisfies the following axioms:

**Red Repl** $!P > P \mid !P$

**Red Match** $[M = M]P > P$.

**Commitment relation.** An *action* is a name $m$ (representing input) or a co-name $\overline{m}$ (representing output) or a distinguished *silent action* $\tau$. Note that actions record only the channel on which the input or the output occurs. The *commitment relation* is written $P \xrightarrow{\alpha} A$, where $P$ is a closed process, $\alpha$ is an action, and $A$ is a closed agent. It is defined by the rules in Tab. 1.

## 3 Relative Addresses and their Composition

We recall here the ideas of [6] that serve as a basis for the authentication mechanisms we are going to introduce. Consider for a while the binary parallel composition as the main operator of the $\pi$-calculus (neither associative nor commutative). Then, build abstract syntax trees of processes as binary trees, called *trees of (sequential) processes*, as follows. Given a process $P$, the nodes of its tree correspond to the occurrences of the parallel operator in $P$, and its leaves are the sequential components of $P$ (roughly, those processes whose top-level operator is a prefix or a summation or a replication). A tree of processes is depicted in Fig. 1.

$$\textit{Comm Out} \qquad \frac{}{\overline{m}\langle M\rangle.P \xrightarrow{\overline{m}} (\nu)\langle M\rangle P}$$

$$\textit{Comm In} \qquad \frac{}{m(x).P \xrightarrow{m} (x)P}$$

$$\textit{Comm Sum 1} \qquad \frac{P \xrightarrow{\alpha} A}{P + Q \xrightarrow{\alpha} A}$$

$$\textit{Comm Sum 2} \qquad \frac{Q \xrightarrow{\alpha} A}{P + Q \xrightarrow{\alpha} A}$$

$$\textit{Comm Par 1} \qquad \frac{P \xrightarrow{\alpha} A}{P|Q \xrightarrow{\alpha} A|Q}$$

$$\textit{Comm Par 2} \qquad \frac{Q \xrightarrow{\alpha} A}{P|Q \xrightarrow{\alpha} P|A}$$

$$\textit{Comm Inter 1} \qquad \frac{P \xrightarrow{m} F \quad Q \xrightarrow{\overline{m}} C}{P|Q \xrightarrow{\tau} F@C}$$

$$\textit{Comm Inter 2} \qquad \frac{P \xrightarrow{\overline{m}} C \quad Q \xrightarrow{m} F}{P|Q \xrightarrow{\tau} C@F}$$

$$\textit{Comm Res} \qquad \frac{P \xrightarrow{\alpha} A \quad \alpha \notin \{m, \overline{m}\}}{(\nu m)P \xrightarrow{\alpha} (\nu m)A}$$

$$\textit{Comm Red} \qquad \frac{P > Q \quad Q \xrightarrow{\alpha} A}{P \xrightarrow{\alpha} A}$$

$$\textit{Comm Struct} \qquad \frac{P \equiv Q \quad Q \xrightarrow{\alpha} A \quad A \equiv A'}{P \xrightarrow{\alpha} A'}$$

Table 1: The commitment relation.

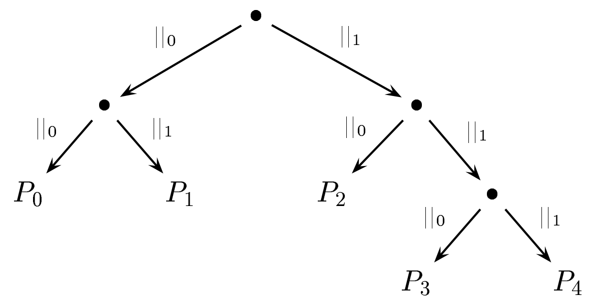


Figure 1: The tree of (sequential) processes of $(P_0|P_1)|(P_2|(P_3|P_4))$.

Assume now that the left (resp. right) branches of a tree of sequential processes denote the left (resp. right) component of parallel compositions, and label their arcs with tag $||_0$ (resp. $||_1$). Therefore, any sequential component in a process is uniquely identified by a string $\vartheta$ over $\{||_0, ||_1\}^*$. The string corresponds to a path from the root, the top-level $|$ of the whole process, to a leaf. Intuitively, $\vartheta$ is the address of the sequential component relative to the root of the binary tree.

Consider now two different sequential processes, say $G$ and $R$, in a tree and call the path between them the *address* of the process $G$ *relative* to the process $R$. This relative address can be decomposed into two parts according to the minimal common predecessor $P$ of $G$ and $R$ in the tree. The relative address is then a string written $\vartheta\bullet\vartheta'$, made of $||_0$'s and $||_1$'s, where $\vartheta$ represents the path from $P$ to $R$, [2] and $\vartheta'$ the path from $P$ to $G$. Let $G$ and $R$ respectively be the processes $P_3$ and $P_1$ of Fig. 1. The address of $P_3$ relative to $P_1$ is then $||_0||_1\bullet||_1||_1||_0$ (read the path upwards from $P_1$ to the root and reverse, then downwards to $P_3$). So to speak, the relative address points back from $R$ to $G$.

Relative addresses can be composed, in order to obtain new relative addresses. For instance, the composition of the relative address $||_1||_0\bullet||_0||_1$ of $P_1$ w.r.t. $P_2$ (in Fig. 1) with the relative address of $P_3$ w.r.t. $P_1$, $||_0||_1\bullet||_1||_1||_0$ is the relative address $||_0\bullet||_1||_0$ of $P_3$ w.r.t. $P_2$.

Below we recall the formal definition of relative addresses and we define their composition. More intuition, the full definitions and the statements of some of their properties are in [6].

**Definition 3.1 (relative addresses)**
*Let $\vartheta_i, \vartheta'_i \in \{||_0, ||_1\}^*$, let $\epsilon$ be the empty string, and let $\oplus$ be the sum modulo 2. Then, the set of* relative addresses*, ranged over by $l$, is*

$$\mathcal{A} = \{\vartheta_0\bullet\vartheta_1 : \vartheta_i = ||_j\vartheta'_i \Rightarrow \vartheta_{i\oplus 1} = ||_{j\oplus 1}\vartheta'_{i\oplus 1}, i, j = 0, 1\}.$$

*We will sometimes omit $\epsilon$ in relative addresses, e.g. we write $\bullet n$ for $\epsilon\bullet\epsilon\ n$.*
*A relative address $l' = \vartheta'\bullet\vartheta$ is* compatible *with $l$, written $l' = l^{-1}$, if and only if $l = \vartheta\bullet\vartheta'$.*

As we said before, we use relative addresses to encode paths between pairs of nodes of binary trees of processes, like the one in Fig. 1. Note that the condition $||_0\vartheta'_0\bullet||_1\vartheta'_1$ (and $||_1\vartheta'_0\bullet||_0\vartheta'_1$) makes it explicit that the two components of the relative address describe the two distinct paths going out from the *same* node in a binary tree. Also, $l' = l^{-1}$ when both refer to the same path, exchanging its source and target.

[2]For technical reasons we take the path from $P$ to $R$ instead of the more natural path from $R$ to $P$.

Address composition is a partial operation, defined only when relative addresses can indeed be composed. We make sure that this is always the case when we apply it. Fig. 2 depicts all the cases in which this happens.

**Definition 3.2 (address composition)** *Address composition $\star : (\mathcal{A} \times \mathcal{A}) \rightharpoonup \mathcal{A}$ is defined by the following three exhaustive cases:*

1. $\vartheta_0{\bullet}\vartheta \star \vartheta_2\vartheta{\bullet}\vartheta_3 = \vartheta_2\vartheta_0{\bullet}\vartheta_3$ *with* $\vartheta_2 \neq \epsilon$
2. $\vartheta_0{\bullet}\vartheta_1\vartheta \star \vartheta{\bullet}\vartheta_3 = \vartheta_0{\bullet}\vartheta_1\vartheta_3$ *with* $\vartheta_1 \neq \epsilon$
3. $\vartheta'\vartheta_0{\bullet}\vartheta \star \vartheta{\bullet}\vartheta'\vartheta_3 = \vartheta_0{\bullet}\vartheta_3$

It is immediate to see that $\star$ has a neutral element (i.e. $l \star \epsilon{\bullet}\epsilon = \epsilon{\bullet}\epsilon \star l = l$), an inverse for each element (i.e. the inverse of $l$ is $l^{-1}$) and that $\star$ is associative (i.e. $(l \star l') \star l'' = l \star (l' \star l'')$), whenever defined.

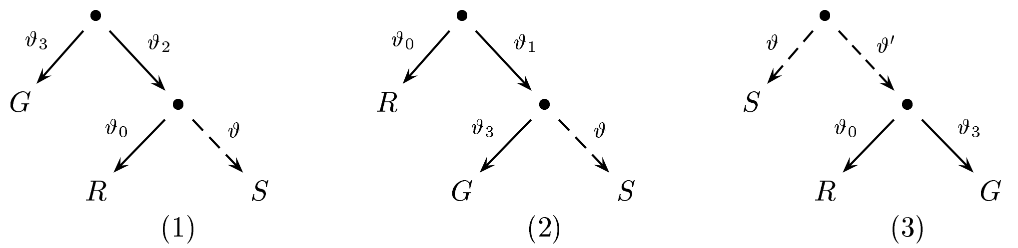


Figure 2: The three possible relative placements of three processes $G$, $S$, and $R$. The result of the composition of the relative addresses of $S$ w.r.t. $R$ and of $G$ w.r.t. $S$ is represented by the solid arrows.

# 4 Proved semantics

Relative addresses can be inductively built while deducing transitions, when a proved semantics is used [9]. In this section, we recall from [10] the proved transition system for the $\pi$-calculus, in which labels of transitions encode (a portion of) their deduction tree. The arrows of the proved transition system are labelled by $\alpha@\vartheta$, where $\alpha$ is an action and $\vartheta$ is a string of $||_0, ||_1$, used to single out the sub-process that actually performed $\alpha$. The rules for proved commitment are in Tab. 2. They are essentially those of the standard transition system except for those involving the parallel operator. Rule *Comm Par 1* (respectively, *Comm Par 2*) adds in the label of its conclusion the tag $||_0$ (respectively, $||_1$) to register that the left (respectively, right) component of a parallel composition is moving. The rules defining the congruence and the reduction relation are indeed the same as before. To recover the standard semantics of the $\pi$-calculus, we only need to erase any occurrence of $\vartheta$ from the labels of transitions.

Note that the information added to labels can be used to inductively build relative addresses. Indeed, the tags $||_i$ are sufficient to recover the parallel structure of a process $P$ because they provide an encoding of the tree of processes of $P$. For instance, suppose that process $\alpha.P$ performs the transition $\alpha@||_0$. Then, we know that the $\alpha$-action was performed by a sequential process on the form $\alpha.P$ in parallel with some process $Q$. Indeed, the whole system had the form $\alpha.P|Q$. More generally, if a process $R$ performs a transition $\alpha@\vartheta$, the path $\vartheta$ in the tree of processes permits to reach the sub-process that performs the $\alpha$ action. Technically, we indicate the sub-process $R$ as $P@\vartheta$, which is inductively selected through the following operator.

$$\frac{}{\overline{m}\langle M\rangle.P \xrightarrow{\overline{m}} (\nu)\langle M\rangle P}\ \textit{Comm Out} \qquad \frac{}{m(x).P \xrightarrow{m} (x)P}\ \textit{Comm In}$$

$$\frac{P \xrightarrow{\alpha @ \vartheta} A}{P + Q \xrightarrow{\alpha @ \vartheta} A}\ \textit{Comm Sum 1} \qquad \frac{Q \xrightarrow{\alpha @ \vartheta} A}{P + Q \xrightarrow{\alpha @ \vartheta} A}\ \textit{Comm Sum 2}$$

$$\frac{P \xrightarrow{\alpha @ \vartheta} A}{P|Q \xrightarrow{\alpha @ ||_0 \vartheta} A|Q}\ \textit{Comm Par 1} \qquad \frac{Q \xrightarrow{\alpha @ \vartheta} A}{P|Q \xrightarrow{\alpha @ ||_1 \vartheta} P|A}\ \textit{Comm Par 2}$$

$$\frac{P \xrightarrow{m @ \vartheta} F \quad Q \xrightarrow{\overline{m} @ \vartheta'} C}{P|Q \xrightarrow{\tau} F @ C}\ \textit{Comm Inter 1} \qquad \frac{P \xrightarrow{\overline{m} @ \vartheta} C \quad Q \xrightarrow{m @ \vartheta'} F}{P|Q \xrightarrow{\tau} C @ F}\ \textit{Comm Inter 2}$$

$$\frac{P \xrightarrow{\alpha @ \vartheta} A \quad \alpha \notin \{m, \overline{m}\}}{(\nu m)P \xrightarrow{\alpha @ \vartheta} (\nu m)A}\ \textit{Comm Res} \qquad \frac{P > Q \quad Q \xrightarrow{\alpha @ \vartheta} A}{P \xrightarrow{\alpha @ \vartheta} A}\ \textit{Comm Red}$$

$$\frac{P \equiv Q \quad Q \xrightarrow{\alpha @ \vartheta} A \quad A \equiv A'}{P \xrightarrow{\alpha @ \vartheta} A'}\ \textit{Comm Struct}$$

Table 2: The proved commitment relation.

**Definition 4.1** *The* localization operator $@\vartheta$ *is defined on processes by induction as follows:*

1. $P\ @\epsilon = P$;
2. $((\nu m)P)\ @\vartheta = (\nu m)(P\ @\vartheta)$;
3. $(P_0 \mid P_1)\ @||_i\vartheta = P_i@\vartheta$, *for* $i = 0, 1$;
4. $([M = N]P)\ @\vartheta = [M = N](P\ @\vartheta)$;

This definition will be helpful at the end of Sections 5 and 8.

Back to Fig. 1, if $P_3$ communicates with $P_1$, the whole process $Sys = (P_0|P_1)|(P_2|(P_3|P_4))$ performs a computation step. The sub-process $P_3$ performing the output is $Sys@||_1||_1||_0$ and the sub-process $P_1$ performing the input is $Sys@||_0||_1$. By putting together the single paths, we obtain the relative address $||_0||_1\bullet||_1||_1||_0$ of $P_3$ relative to $P_1$.

# 5 Partner authentication

We now introduce our first authentication mechanism. At run-time, it will guarantee each principal to engage an entire run session with the same partners playing the same roles. We heavily exploit the proved semantics reported in the previous section. We essentially bind sensitive input and output communications to relative addresses. More precisely, channels may have a relative address as index, and assume the form $c_l$. Now, our semantics will ensure that $P$ communicates with $Q$ on $c_l$ if and only if the relative address of $P$ w.r.t. $Q$ is indeed $l$ (and that of $Q$ w.r.t. $P$ is $l^{-1}$). Notably, even if another process $R \neq Q$ possesses the channel $c_l$, $R$ cannot use it to communicate with $P$. Consequently, a hostile process can never interfere with $P$ and $Q$ while they communicate, not even eavesdrop the exchanged messages.

By using these "located" channels, processes may have some control on their partners. But often a process $P$ is willing to communicate with several processes, usually through one or very few channels. So, we shall also index a channel with a variable $\lambda$ to be instantiated. Suppose that process $P$, playing for instance the role of sender or of initiator, wants to communicate with a process $S$ (e.g. the server), that $P$ does not know the relative address of $S$, say $l$, and finally that the involved channel is $c_\lambda$. Then, during the first communication of $P$ with $S$, $\lambda$ will be instantiated within $P$ by $l$ (recall that our proved operational semantics indeed computes $l$). Symmetrically for $S$, if it uses the same channel $c$ (with a different variable $\lambda'$, i.e., $S$ uses $c_{\lambda'}$). In a sense, this is the case of anonymous communication. Note however that $S$ may as well use an already located channel $c_{l'}$: the communication occurs only if $l^{-1} = l'$. From that point on, $P$ and $S$ will keep communicating in the same roles for the entire session, using their, now known, relative addresses.

Thus, we extend the names that occur in processes by indexing them with a *location*, defined to be either a relative address $l = \vartheta\bullet\vartheta'$ or a variable $\lambda$ to be instantiated by a relative address. Formally,

**Definition 5.1** *Let* $\Lambda \ni \lambda, \lambda', \ldots, \lambda_0, \ldots$ *be a countable set of (address) variables, and let* $\mathrm{Loc} = \Lambda \cup \mathcal{A} \ni t$. *Then,* $\mathcal{N}_{\mathrm{Loc}} = \{m_t \mid m \in \mathcal{N}, t \in \mathrm{Loc}\}$ *is the set of* located channels.
*Usually, the empty location* $t = \epsilon\bullet\epsilon$ *is omitted.*

The rules defining the congruence and the reduction relation are the same as before, apart from the obvious substitution of located names for names. The rules for the new commitment relation are in Tab. 3, where we omit the symmetric rules for communication. The rules for parallel composition are in the proved style,

$$\mathit{Comm\ Out}\quad \frac{}{\overline{m_t}\langle M\rangle.P \xrightarrow[t]{\overline{m}} (\nu)\langle M\rangle P} \qquad\qquad \mathit{Comm\ In}\quad \frac{}{m_t(x).P \xrightarrow[t]{m} (x)P}$$

$$\mathit{Comm\ Sum}\ 1\quad \frac{P \xrightarrow[t]{\alpha@\vartheta} A}{P+Q \xrightarrow[t]{\alpha@\vartheta} A+Q} \qquad\qquad \mathit{Comm\ Par}\ 1\quad \frac{P \xrightarrow[t]{\alpha@\vartheta} A}{P|Q \xrightarrow[t]{\alpha@||_0\vartheta} A|Q}$$

$$\mathit{Comm\ Inter}_L\ 1\quad \frac{P \xrightarrow[l]{m@\vartheta} F \quad Q \xrightarrow[l']{\overline{m}@\vartheta'} C}{P|Q \xrightarrow{\tau} F@C} \quad \text{if} \begin{cases} (l\ = ||_0\vartheta\bullet||_1\vartheta' \ \vee\ l = \epsilon\bullet\epsilon) \ \wedge \\ (l'\ = ||_1\vartheta'\bullet||_0\vartheta \ \vee\ l' = \epsilon\bullet\epsilon) \end{cases}$$

$$\mathit{Comm\ Inter}_{L,\Lambda}\ 1\quad \frac{P \xrightarrow[l]{m@\vartheta} F \quad Q \xrightarrow[\lambda']{\overline{m}@\vartheta'} C}{P|Q \xrightarrow{\tau} F@C\{\!|l'/\lambda'|\!\}_{@\vartheta'}} \quad \begin{cases} \text{if } (l = ||_0\vartheta\bullet||_1\vartheta' \ \vee l = \epsilon\bullet\epsilon) \\ \text{where } l' = ||_1\vartheta'\bullet||_0\vartheta \end{cases}$$

$$\mathit{Comm\ Inter}_{\Lambda}\ 1\quad \frac{P \xrightarrow[\lambda]{m@\vartheta} F \quad Q \xrightarrow[\lambda']{\overline{m}@\vartheta'} C}{P|Q \xrightarrow{\tau} F\{\!|l/\lambda|\!\}_{@\vartheta}@C\{\!|l^{-1}/\lambda'|\!\}_{@\vartheta'}} \quad \text{where } l = ||_0\vartheta\bullet||_1\vartheta'$$

$$\mathit{Comm\ Res}\quad \frac{P \xrightarrow[t]{\alpha@\vartheta} A \quad \alpha \notin \{m, \overline{m}\}}{(\nu m)P \xrightarrow[t]{\alpha@\vartheta} (\nu m)A} \qquad\qquad \mathit{Comm\ Red}\quad \frac{P > Q \quad Q \xrightarrow[t]{\alpha@\vartheta} A}{P \xrightarrow[t]{\alpha@\vartheta} A}$$

$$\mathit{Comm\ Struct}\quad \frac{P \equiv Q \quad Q \xrightarrow[t]{\alpha@\vartheta} A \quad A \equiv A'}{P \xrightarrow[t]{\alpha@\vartheta} A'}$$

Table 3: The proved located commitment relation.

recording which component (left or right) of the process is moving. There, some of the arrows are annotated also with a location. For each I/O action rule, the location $t$ of the channel involved is recorded under the arrow, and it is preserved by all non communication rules and discarded by communications. This location $t$ is used in the premises of communication rules, to establish the relative addresses of one process with respect to the other. In fact, if the first process (the receiver) performs an input $m_@\vartheta$ and the second process (the sender) performs the complementary output action $\overline{m}_@\vartheta'$, then the relative address of the sender with respect to the receiver is $||_0\vartheta\bullet||_1\vartheta'$. (The additional $||_0$ and $||_1$ record that the two processes are the left and the right partners in the communication.)

There are three different rules, up to symmetries, for every communication between two processes, say $P$ and $Q$. We comment on them, and we intuitively relate them with the three possible situations in which $P$ and $Q$ may be.

$Comm\ Inter_L\ 1$ $P$ wants to receive from a process located at $l$, and $Q$ wants to send to a process located at $l'$. For the communication to happen the relative addresses of $Q$ w.r.t. $P$ and of $P$ w.r.t. $Q$ (the path established by the communication) should coincide with these locations that should be compatible, i.e. $l' = l^{-1}$, as the side conditions require. This situation reflects the fact that $P$ and $Q$ "know each other", possibly because they have previously established their connection and are session partners.

Note that when $l$ and $l'$ are $\epsilon\bullet\epsilon$, we recover the "non-located" communication of the standard $\pi$-calculus.

$Comm\ Inter_{L,\Lambda}\ 1$ $P$ wants to receive from a process located at $l$, while $Q$ is willing to send a message to any process (it sends on a channel with location variable $\lambda'$). The communication is successful only if $l$ coincides with the path established by the communication, i.e. if $l$ coincides with the relative address of $Q$ w.r.t. $P$: $Q$ is indeed the process from which $P$ wants to receive. After the communication, $\lambda'$ will be suitably bound to $l^{-1}$ (the relative address of $P$ w.r.t. $Q$) within $Q$, so that now $P$ and $Q$ "know each other".

If $l$ is the empty location, then the communication is always successful and $\lambda'$ will still be replaced by the relative address of $P$ w.r.t. $Q$.

$Comm\ Inter_\Lambda\ 1$ $P$ and $Q$ do not "know each other" and are willing to communicate with any partner. So, they exchange their relative addresses, as established while deducing the premises. Intuitively, $P$ and $Q$ are performing their initial synchronization. So their two variables are replaced with the relative address of $Q$ w.r.t. $P$ and vice-versa ($l$ and $l'$, respectively).

Variables are not located. Consequently, when a located channel $c_l$ is communicated, it becomes a "free" channel: the location $l$ is lost. The index to $c$ becomes $\epsilon\bullet\epsilon$ if so it was; otherwise we get $c_\lambda$ (with $\lambda$ not occurring in the process). Formally, we have the following.

**Definition 5.2** *Let $F = (y)P$ be a abstraction and $C = (\nu\tilde{n})\langle c_l\rangle Q$ be a concretion. Then, their interaction is*

$$F@C = (\nu\tilde{n})(P\{c_t/y\}|Q),\ \textit{where } t = \begin{cases} \epsilon\bullet\epsilon & \textit{if } l = \epsilon\bullet\epsilon \\ \lambda & \textit{otherwise (with } \lambda \textit{ not occurring in } Q) \end{cases}$$

*Symmetrically for $C@F$.*

Using the above definition for communication makes it easier to use a channel in a multiplexing way. Suppose that a process $P$ is committed to communicate on a channel $c_l$ with a process $Q$. Also, assume that

$P$ sends $c_l$ to a third party $R$, that receives it as $c_\lambda$. The "same" channel $c$ can now be used for further communications between $P$ and $Q$ (in this case located at $l$), as well as for communications between $R$ and some other process (after $\lambda$ has been suitably instantiated).

In the rules for commitment we use the particular kind of substitution $\{\!|l/\lambda|\!\}_{@\vartheta}$, called selective routed substitution. It uses a particular substitution $\{\!|l/\lambda|\!\}$, called routed, to be applied only to the sub-process located at $\vartheta$. The routed substitution takes into account the parallel structure of the process. For that it updates the relative addresses $l$ while traversing the tree of processes. For instance, to substitute $\vartheta\bullet\vartheta'$ for $\lambda$ in $P_0|P_1$ requires to substitute $||_i\bullet\epsilon \star \vartheta\bullet\vartheta'$ for $\lambda$ in each component $P_i$.

**Definition 5.3** *The* location routed substitution $\{\!|l/\lambda|\!\}$ *is defined by induction as follows:*

1. $m_{\lambda'}\{\!|l/\lambda|\!\} = \begin{cases} m_{\lambda'} & \lambda \neq \lambda' \\ m_l & \textit{otherwise} \end{cases}$
2. $(\overline{m_\lambda}\langle M\rangle.P)\{\!|l/\lambda|\!\} = \overline{m_\lambda}\{\!|l/\lambda|\!\}\langle M\{\!|l/\lambda|\!\}\rangle.P\{\!|l/\lambda|\!\}$
3. $(m_\lambda(x).P)\{\!|l/\lambda|\!\} = m_\lambda\{\!|l/\lambda|\!\}(x).P\{\!|l/\lambda|\!\}$
4. $(P_0|P_1)\{\!|l/\lambda|\!\} = P_0\{\!|\,||_0\bullet \star l/\lambda|\!\}|P_1\{\!|\,||_1\bullet \star l/\lambda|\!\}$;
5. $\mathbf{0}\{\!|l/\lambda|\!\} = \mathbf{0}$
6. $(P+Q)\{\!|l/\lambda|\!\} = P\{\!|l/\lambda|\!\} + Q\{\!|l/\lambda|\!\}$
7. $[M = N]P\{\!|l/\lambda|\!\} = [M\{\!|l/\lambda|\!\} = N\{\!|l/\lambda|\!\}](P\{\!|l/\lambda|\!\})$

There is no case for $!P$ in the above definition, i.e. the routed substitution may be applied only to $P$, after that $!P$ has been reduced to $P\ |!P$. This amounts to saying that we are also considering processes that inside have routed substitutions not fully performed. Consequently, the target of the transitions in Tab. 3 may contain expressions on the form $!P\{\!|l/\lambda|\!\}$. In order not to burden too heavily our notation, we shall still use $P$ and $A$ for processes, abstractions and concretions with substitutions not yet performed.

We use the above definition to implement a selective substitution that works on the sub-process of a whole term reachable through $\vartheta$.

**Definition 5.4** *Let* $\vartheta \in \{||_0, ||_1\}^*$. *Then, the* selective routed substitution $P\{\!|l/\lambda|\!\}_{@\vartheta}$ *is defined by induction as*

- $(P_0|P_1)\{\!|l/\lambda|\!\}_{@\vartheta} = \begin{cases} P_0\{\!|l/\lambda|\!\}_{@\vartheta'}|P_1 & \textit{if } \vartheta = ||_0\vartheta' \\ P_0|P_1\{\!|l/\lambda|\!\}_{@\vartheta'} & \textit{if } \vartheta = ||_1\vartheta' \end{cases}$
- $P\{\!|l/\lambda|\!\}_{@\epsilon} = P\{\!|l/\lambda|\!\}$

The very fact that channels have indexes, which may also be instantiated, guarantees that two partners can establish a connection that will remain stable along a whole session. Indeed, let $P_1 = (Q|R)|P$, where $Q = c_{||_0\bullet||_1}(y).Q'$ (note that $||_0\bullet||_1$ is the address of $R$ relative to $Q$). Then, it is immediate verifying that $Q$ accepts inputs on $c$ only if they come from $R$; of course this property remains true for all inputs of $Q$ along $c_{||_0\bullet||_1}$. Symmetrically for the process $P_2 = (\overline{c}_{||_0\bullet||_1}\langle M\rangle.Q'|R)|P$.

**Example 5.5** We now illustrate our partner authentication mechanism, through some simple examples. First of all, consider a single message exchange between Alice, $A$ and Bob, $B$:

$$\begin{array}{rcl} A & = & \overline{c}_\lambda\langle M\rangle.A' \\ B & = & c_{\lambda'}(x).B' \\ P & = & A|B \end{array}$$

where $\lambda$ and $\lambda'$ are two location variables. After the message exchange, the semantic rule $Comm\ Inter_\Lambda$ 1 instantiates $\lambda$ and $\lambda'$ in $A'$ and $B'$, respectively, with the address of $A$ relative to $B$ (i.e., $||_1\bullet||_0$) and of $B$ relative to $A$ (i.e., $||_0\bullet||_1$), respectively. ∎

Intuitively, the instantiation of a $\lambda$ with an $l$ represents a secure declaration of the identity of a process through its location $l$, that cannot be manipulated even by malicious parties. As we will see later on, located actions also give some security guarantees on the subsequent message exchanges.

**Example (cont'd)** Consider now the following protocol (recall that $d_{\epsilon\bullet\epsilon}$ is simply written as $d$; we shall comment below on the use of channel $d$, which occurs located in $A'$ but not in $B'$).

$$\begin{array}{rcl} A' & = & \overline{c}_\lambda\langle M_A\rangle.d_\lambda(y) \\ B' & = & c_{\lambda'}(x).\overline{d}\langle M_B\rangle \\ P' & = & A'|B' \end{array}$$

Here, Bob sends a message to Alice after the reception of $M_A$. Note that Alice is requiring that the second message comes from the same process to which she sent $M_A$. Since, after the first message exchange, the variable $\lambda$ is instantiated to the address of $B'$ relative to $A'$, our semantics guarantees authentication of the second communication with respect to the (secure) identity declaration of $\lambda$: Alice is assured that the second message will be sent from the same process that received the first one. In order to illustrate this important point we add another process $C' = \overline{d}\langle M_C\rangle + d(x).D$ which tries to communicate over channel $d$. The process $P'|C'$ has the following steps:

$$(A'|B')|C' \xrightarrow{\tau} (\ d_{||_0\bullet||_1}(y)\ \mid\ \overline{d}\langle M_B\rangle\ )\ \mid\ (\overline{d}\langle M_C\rangle + d(x).D)$$

Since the address of $C'$ relative to $A'$ is $||_0||_0\bullet||_1 \neq ||_0\bullet||_1$, then either (the residual of) $A'$ receives from and only from (the residual of) $B'$ or $B'$ and $C'$ communicate. In the first case we have:

$$(\ d_{||_0\bullet||_1}(y)\ \mid\ \overline{d}\langle M_B\rangle\ )\ \mid\ (\overline{d}\langle M_C\rangle + d(x).D) \xrightarrow{\tau} (\ \mathbf{0}\ \mid\ \mathbf{0}\ )\ \mid\ (\overline{d}\langle M_C\rangle + d(x).D)$$

In the second case we have:

$$(\ d_{||_0\bullet||_1}(y)\ \mid\ \overline{d}\langle M_B\rangle\ )\ \mid\ (\overline{d}\langle M_C\rangle + d(x).D) \xrightarrow{\tau} (\ d_{||_0\bullet||_1}(y)\ \mid\ \mathbf{0}\ )\ \mid\ D[M_B/x]$$

∎

In the example above, the same channel $d$ is used in two different ways: it is "located" for Alice and Bob; alternatively, it is "free" for $B'$ and $C'$. In Example 5.9, we shall see that the same channel can be even used in a multiplexing fashion: two pairs of processes can interleave their communications, still presenting the property of being engaged with the same process along the entire session.

In the next example we consider the situation where the channel $d$ is located and used to output a message $M$. This usage of channels also guarantees a sort of secrecy of $M$.

**Example 5.6** We have seen in the previous example that message $M_B$ was intercepted by $C'$, thus violating its secrecy. Consider now the following protocol:

$$\begin{array}{rcl} A'' & = & \overline{c}_\lambda\langle M_A\rangle.d(y) \\ B'' & = & c_{\lambda'}(x).\overline{d}_{\lambda'}\langle M_B\rangle \\ P'' & = & A''|B'' \end{array}$$

Here, Bob is requiring that the message $M_B$ is received by the same user that sent the first message. In this case Bob obtains a form of secrecy: he is assured that $M_B$ will be read by the process identified by $\lambda'$, i.e., the process that sent message $M_A$. Indeed, any new process $C'' = d(w)$ fails to read $M_B$. We have that

$$(A''|B'')|C'' \xrightarrow{\tau} (\ d(y)\ \mid\ \overline{d}_{||_1\bullet||_0}\langle M_B\rangle\ )\ \mid\ d(w)$$

but the address of $C''$ relative to $B''$ is $||_0||_1\bullet||_1 \neq ||_1\bullet||_0$. ■

We have seen that locating inputs and outputs corresponds to guaranteeing authentication and secrecy of the communication, respectively. We can summarize these two concepts as follows. In a message exchange and with respect to an address $l$, a process obtains

**Partner authentication** whenever it receives the message from the process reachable at $l$, only.

**Secrecy** whenever only the process reachable at $l$ will receive the message.

We now state the above more precisely, exploiting Def. 4.1. We need the notion of context with two holes, written $\mathcal{C}[-,-]$.

**Theorem 5.7 (authentication)** *Let $\hat{Q} = \mathcal{C}[R,S]$, $R = c_l(x).P_0$ and $S = \overline{c}_t\langle M\rangle.P_1$. Then, $\hat{Q} > Q \xrightarrow{\tau} \mathcal{C}'[P_0\{M/x\}, P_1]$, with $Q@\vartheta\vartheta_0 = R$ and $Q@\vartheta\vartheta_1 = S$, if and only if $l = \vartheta_0 \bullet \vartheta_1$.*

PROOF. By inspection on the rules used to deduce the transition; in particular consider the side conditions of rules $Comm\ Inter_L\ 1$ (where $t = l^{-1}$) and $Comm\ Inter_{L,\Lambda}\ 1$ (where $t = \lambda$). ■

**Theorem 5.8 (secrecy)** *Let $\hat{Q} = \mathcal{C}[R,S]$, $R = \overline{c}_l\langle M\rangle.P_0$ and $S = c_t(x).P_1$. Then, $\hat{Q} > Q \xrightarrow{\tau} \mathcal{C}'[P_0, P_1\{M/x\}]$, with $Q@\vartheta\vartheta_0 = R$ and $Q@\vartheta\vartheta_1 = S$, if and only if $l = \vartheta_0 \bullet \vartheta_1$.*

PROOF. Analogous to the previous proof. ■

The next example illustrates how locating both inputs and outputs guarantees a permanent hooking between two parties and allows to model multiple sessions quite easily.

**Example 5.9** Consider the following processes:

$$\begin{array}{rcl} \tilde{A} & = & \overline{c}_\lambda\langle M_A\rangle.d_\lambda(y) \\ \tilde{B} & = & c_{\lambda'}(x).\overline{d}_{\lambda'}\langle M_B\rangle \\ \tilde{P} & = & \tilde{A}|\tilde{B} \end{array}$$

Indeed both Alice and Bob are assured that the second message (and also all the subsequent messages sent and received on located channels) will be sent/received by the user that interacted in the first message exchange. Hence, the two users, after the first communication are permanently hooked together. This time a third user $\tilde{C} = c_{\lambda'}(x).\overline{d}_{\lambda'}\langle M_C\rangle$, is indeed able to take the place of $\tilde{B}$ in the communication but only if it starts the session with $\tilde{A}$. Instead, it can never communicate with $\tilde{A}$ after the first message exchange between $\tilde{A}$ and $\tilde{B}$ occurred.

We now model multiple sessions. Consider $\tilde{P}_{multi} = !(\nu M_A)\tilde{A}|!(\nu M_B)\tilde{B}$, where an unbounded number of instances of Alice and Bob are present, each with a different fresh message. Consider now two instances of $\tilde{A}$ sending their first messages to two instances of $\tilde{B}$, i.e, two parallel sessions:

$$\begin{array}{lll}
\tilde{P}_{multi} & \xrightarrow{\tau} & (\ d_{||_0||_0\bullet||_1||_0}(y)\ \mid\ !(\nu M_A)\tilde{A}\ )\ \mid\ (\ \overline{d}_{||_1||_0\bullet||_0||_0}\langle M_B^1\rangle\ \mid\ !(\nu M_B)\tilde{B}\ ) \\
 & \xrightarrow{\tau} & (\ d_{||_0||_0\bullet||_1||_0}(y)\ \mid\ (\ d_{||_0||_1||_0\bullet||_1||_1||_0}(y)\ \mid\ !(\nu M_A)\tilde{A}\ )\ )\ \mid \\
 & & (\ \overline{d}_{||_1||_0\bullet||_0||_0}\langle M_B^1\rangle\ \mid\ (\ \overline{d}_{||_1||_1||_0\bullet||_0||_1||_0}\langle M_B^2\rangle\ \mid\ !(\nu M_B)\tilde{B}\ )\ )
\end{array}$$

Note that after the first message exchange the partners in each session are permanently hooked: the second message is always sent to the correct party, the one who initiated the session. As a consequence, no replay of messages is possible among different sessions, also in the presence of a malicious party. ■

# 6 The Spi-Calculus

**Syntax.** In this section we briefly recall, often also literally, the spi-calculus [3], in its monadic version from [1]. This calculus is an extension of the $\pi$-calculus, introduced for the description and the analysis of cryptographic protocols. A first difference with the $\pi$-calculus is that the spi-calculus has no summation operator $+$. Also, terms can be structured as pairs $(M, N)$, successors of terms $suc(M)$ and encryptions $\{M_1, \ldots, M_k\}_N$. The last term above represents the ciphertext obtained by encrypting $M_1, \ldots, M_k$ under the key $N$, using a shared-key cryptosystem such as DES [27].

**Definition 6.1** *Terms and processes are defined according to the following BNF-like grammars.*

$$\begin{array}{ll}
L, M, N ::= & terms \\
\quad n & \quad names \\
\quad x & \quad variables \\
\qquad (M, N) & \quad pair \\
\qquad 0 & \quad zero \\
\qquad suc(M) & \quad successor \\
\qquad \{M_1, \ldots, M_k\}_N & \quad shared-key\ encryption
\end{array}$$

$$\begin{array}{ll}
P, Q, R ::= & processes \\
\quad \mathbf{0} & \quad nil \\
\quad \pi.P & \quad prefix \\
\quad P|P & \quad parallel\ composition \\
\quad (\nu m)P & \quad restriction \\
\quad [M = N]P & \quad matching \\
\quad !P & \quad replication \\
\qquad let\ (x, y) = M\ in\ P & \quad pair\ splitting \\
\qquad case\ M\ of\ 0 : P\ suc(x) : Q & \quad integer\ case \\
\qquad case\ L\ of\ \{x_1, \ldots, x_k\}_N\ in\ P & \quad shared{-}key\ decryption
\end{array}$$

*where $\pi$ may either be $M(x)$ or $\overline{M}\langle N\rangle$.*[3]

Most of the process constructs are the same of $\pi$-calculus. The new ones decompose terms:

- The process $let\ (x, y) = M\ in\ P$ behaves as $P[M_0/x, M_1/y]$ if $M = (M_0, M_1)$ and it is stuck if $M$ is not a pair;
- The process $case\ M\ of\ 0 : P\ suc(x) : Q$ behaves as $P$ if $M$ is 0, as $Q[N/x]$ if $M = suc(N)$ and it is stuck otherwise;
- The process $case\ L\ of\ \{x_1, \dots, x_k\}_N\ in\ P$ attempts to decrypt $L$ with the key $N$; if $L$ has the form $\{M_1, \dots, M_k\}_N$, then the process behaves as $P[M_1/x_1, \dots, M_k/x_k]$, and otherwise is stuck.

The structural congruence and the operational semantics for commitment are exactly the same of the $\pi$-calculus given in Table 1. Some new reductions rules are instead needed.

**Red Split** $let\ (x, y) = (M, N)\ in\ P > P[M/x, N/y]$

**Red Zero** $case\ 0\ of\ 0 : P\ suc(x) : Q > P$

**Red Suc** $case\ suc(M)\ of\ 0 : P\ suc(x) : Q > Q[M/x]$

**Red Decrypt** $case\ \{M_1, \dots, M_k\}_N\ of\ \{x_1, \dots, x_k\}_N\ in\ P > P[M_1/x_1, \dots, M_k/x_k]$.

## 7 Names of the spi-calculus handled locally

To introduce our second authentication mechanism, we need to further exploit the ideas contained in [6], where the relative addresses, introduced in Section 3, are used to handle names locally to sequential processes in an operational manner. The space of names of a whole process is then partitioned into local environments associated each with its sequential sub-processes.

To avoid global management of names, we have to solve two problems. Names have to be declared *locally* and to be brand-new in that local environment. Furthermore, when a name is exported to other local environments via communications or by applying a reduction rule, we must guarantee that there are *no clashes* involving the other names around. A purely mechanical way of doing that is in [6].

For the sake of simplicity, instead of recalling also the mechanism for generating fresh names, here we assume that a name is fresh, whenever needed, and we shall recall that by a side condition.

As for keeping names distinct, consider two different sequential processes, say $G$ and $R$, that have two syntactically equal names, say $n$. Suppose now that $G$ sends $n$ to $R$. To distinguish between the two different instances of $n$ in the local environment of $R$, the name generated by $G$ will be received enriched with the *address* of $G$ *relative* to $R$, which points back from $R$ to the local environment of $G$.

A slightly more complex situation arises when a process receives a name and sends it to another process. The name must arrive at the new receiver with the address of the generator (*not* of the sender) relative to the new receiver. Consider again Fig. 1, where $P_1$ sends to $P_2$ a name generated by $P_3$. The rules (in Tab. 4) for communication use address composition to determine the address of $P_3$ relative to $P_2$, by composing the address of the message (recording the address of $P_3$ w.r.t. $P_1$) with the relative address of $P_1$ w.r.t. $P_2$.

We carry the localized semantics of the $\pi$-calculus of [6] on the monadic spi-calculus. First of all, we introduce the new set of localized names, that are names prefixed with relative addresses.

[3]Although $M$ is an arbitrary term, we only consider it to be a name or a variable (to be instantiated to a name), because these are the only useful cases (see [3]).

**Definition 7.1** *Let $\mathcal{N}' = (\mathcal{A} \cdot \mathcal{N})$ be the set of* localized names*, where $\mathcal{N}$ is the set of* standard names *and "$\cdot$" is the operator of language concatenation.*

For simplicity, we assume $r, s, u, \dots$, possibly indexed, to range over both $\mathcal{N}'$ and $\mathcal{N}$ and, unless necessary, we do not syntactically distinguish localized terms from terms, i.e. terms prefixed with relative addresses like $\vartheta{\bullet}\vartheta' M$, from those not prefixed.

As we said above, we do not recall how the mechanism of [6] generates fresh names whenever needed: here we simply assume them fresh. However, we require that restricted names are always localized, i.e. they occur in a declaration as $(\nu{\bullet}n)$. Technically, this is achieved by transforming a process $P$ with $(\nu n)$ into a new process, obtained by replacing each sub-process of P on the form $(\nu n)Q$ with the process $(\nu{\bullet}n)Q\{\!|{\bullet}n/n|\!\}_s$ (the substitution $\{\!|/|\!\}_s$ is in Def. 7.3).

When a term $M$ is exported from a process, say $P$, to another, say $Q$, it is necessary to compose the relative address prefixing $M$ with the relative address of $P$ w.r.t. $Q$. This composition is performed by the term address composition, that extends address composition in Def. 3.2. Applied to a relative address and to a localized term, it returns an updated localized term.

**Definition 7.2** Term address composition $\star^T$ *is defined as*

$$\vartheta{\bullet}\vartheta' \star^T \vartheta_0{\bullet}\vartheta_1 M = (\vartheta{\bullet}\vartheta' \star \vartheta_0{\bullet}\vartheta_1)M.$$

*We say that $\vartheta{\bullet}\vartheta' \star^T \vartheta_0{\bullet}\vartheta_1 M$, is the term $\vartheta_0{\bullet}\vartheta_1 M$* exported *to the relative address $\vartheta{\bullet}\vartheta'$.*

Names in $\mathcal{N}$, variables and natural numbers are not prefixed with relative addresses and are insensitive to address composition:

$$\forall z \in \mathcal{N} \cup Nat \cup \mathcal{V} :\ \vartheta{\bullet}\vartheta' \star z \equiv z$$

We now explain how our semantics deals with terms. First, note that the operator $\star^T$ considers the compound term $M$ as a whole, as it does not distribute address composition over the sub-terms of $M$. Now, consider the encryption term $\{M\}_K$. It is an atomic entity, and so it is handled as it were a *new* name, local to the process that encrypts $M$, say $P$. The two localized terms $M$ and $K$ are frozen just like they were at encryption time, and their own relative addresses are not changed when the encrypted message is sent through the net; again, these relative addresses *cannot* be updated. Since $\{M\}_K$ is atomic, its relative address, say $\vartheta_0{\bullet}\vartheta_1$, will always point to the process $P$ that made the encryption. (Technically, $M$ and $K$ are frozen like they were in the process $S$ containing all the restrictions on the names used in the encryption.) In this way, when decrypting $\{M\}_K$ the semantic rules recover the correct addresses for $M$ and $K$ by simply composing the actual address of $\{M\}_K$, $\vartheta_0{\bullet}\vartheta_1$, with the (frozen) relative addresses of $M$ and $K$, respectively. The same management described above is used for successor and pairs. In the first case, the term $M$ is frozen in $suc(M)$, while the terms $M$ and $N$ are frozen in $(M, N)$.

Also the routed substitution of Def. 5.3 is extended to deal both with terms and with processes in the spi-calculus; it distributes to each sub-term or sub-process (note that below the term $N$ cannot be a variable).

**Definition 7.3** *The* spi routed substitution $\{\!|N/x|\!\}_s$ *is defined by induction as follows on*

**terms**

1. $r\{\!|N/x|\!\}_s = r$, *with* $r \in \mathcal{N} \cup \mathcal{N}'$;
2. $z\{\!|N/x|\!\}_s = \begin{cases} z & x \neq z \\ N & \textit{otherwise;} \end{cases}$

3. $0\{\!|N/x|\!\}_s = 0$;
4. $suc(M)\{\!|N/x|\!\}_s = suc(M\{\!|N/x|\!\}_s)$;
5. $(M_1, M_2)\{\!|N/x|\!\}_s = (M_1\{\!|N/x|\!\}_s, M_2\{\!|N/x|\!\}_s)$;
6. $\{M_1, \ldots, M_k\}_{M_0}\{\!|N/x|\!\}_s = \{M_1\{\!|N/x|\!\}_s, \ldots, M_k\{\!|N/x|\!\}_s\}_{M_0\{\!|N/x|\!\}_s}$.

**processes**

1. $\mathbf{0}\{\!|N/x|\!\}_s = \mathbf{0}$;
2. $(\overline{r}\langle M\rangle.P)\{\!|N/x|\!\}_s = \overline{r}\{\!|N/x|\!\}_s\langle M\{\!|N/x|\!\}_s\rangle.P\{\!|N/x|\!\}_s$
3. $(r(y).P)\{\!|N/x|\!\}_s = \begin{cases} r\{\!|N/x|\!\}_s(y).P & x = y \\ r\{\!|N/x|\!\}_s(y).P\{\!|N/x|\!\}_s & \textit{otherwise;} \end{cases}$
4. $(P \mid Q)\{\!|N/x|\!\}_s = P\{\!|\|_0\bullet \star^T N/x|\!\}_s \mid Q\{\!|\|_1\bullet \star^T N/x|\!\}_s$,
5. $((\nu\vartheta_\bullet\vartheta' n)P)\{\!|N/x|\!\}_s =$
$\begin{cases} (\nu\vartheta_\bullet\vartheta' m)(P\{\!|\vartheta_\bullet\vartheta' m/\vartheta_\bullet\vartheta' n|\!\}_s)\{\!|N/x|\!\}_s & \vartheta_\bullet\vartheta' n = N \textit{ and } \vartheta_\bullet\vartheta' m \textit{ fresh in } P \\ (\nu\vartheta_\bullet\vartheta' n)P\{\!|N/x|\!\}_s & \textit{otherwise;} \end{cases}$
6. $([M_1 = M_2]P)\{\!|N/x|\!\}_s = [M_1\{\!|N/x|\!\}_s = M_2\{\!|N/x|\!\}]P\{\!|N/x|\!\}_s$;
7. $(let\ (y_1, y_2) = M\ in\ P)\{\!|N/x|\!\}_s =$
$\begin{cases} let\ (y_1, y_2) = M\{\!|N/x|\!\}_s\ in\ P & x = y_i \textit{ for some } i \\ let\ (y_1, y_2) = M\{\!|N/x|\!\}_s\ in\ P\{\!|N/x|\!\}_s & \textit{otherwise;} \end{cases}$
8. $(case\ M\ of\ 0 : P\ suc(y) : Q)\{\!|N/x|\!\}_s =$
$\begin{cases} case\ M\{\!|N/x|\!\}_s\ of\ 0 : P\{\!|N/x|\!\}_s\ suc(y) : Q & x = y \\ case\ M\{\!|N/x|\!\}_s\ of\ 0 : P\{\!|N/x|\!\}_s\ suc(y) : Q\{\!|N/x|\!\}_s & \textit{otherwise;} \end{cases}$
9. $(case\ L\ of\ \{y_1, \ldots, y_k\}_M\ in\ P)\{\!|N/x|\!\}_s =$
$\begin{cases} case\ L\{\!|N/x|\!\}_s\ of\ \{x_1, \ldots, x_k\}_{M\{\!|N/x|\!\}_s}\ in\ P & x = y_i \textit{ for some } i \\ case\ L\{\!|N/x|\!\}_s\ of\ \{x_1, \ldots, x_k\}_{M\{\!|N/x|\!\}_s}\ in\ P\{\!|N/x|\!\}_s & \textit{otherwise.} \end{cases}$

Now, the selective routed substitution for the spi-calculus is exactly as in Def. 5.4.

**Localized Congruence.** The rules for the structural congruence require some changes to accommodate localized names.

1. $(\nu r)P_1|P_2 \equiv (\nu_\bullet\|_0 \star r)(P_1|P_2)$;
2. $P_1|(\nu r)P_2 \equiv (\nu_\bullet\|_1 \star r)(P_1|P_2)$;
3. $(\nu r)\langle M\rangle P_1 \mid P_2 \equiv (\nu_\bullet\|_0 \star r)\langle\bullet\|_0 \star^T M\rangle(P_1 \mid P_2)$;
4. $P_1 \mid (\nu r)\langle M\rangle P_2 \equiv (\nu_\bullet\|_1 \star r)\langle\bullet\|_1 \star^T M\rangle(P_1 \mid P_2)$.

Note that no $\alpha$-conversion is needed: each binding occurrence of the name $r$ in $P_1$ ($P_2$, resp.) is replaced by $\|_0 \star r$ ($\|_1 \star r$, resp.) which is different from any name $s$ in $P_1$ ($P_2$, resp.) , because of the properties of address composition $\star$.

**Localized Reduction Relation.** We add the following reduction rules to those for matching and replication.

**Red Split** $let\ (x, y) = \vartheta_0 \bullet \vartheta_1 (M, N)\ in\ P > P\{\!|\vartheta_0 \bullet \vartheta_1 \star^T M/x|\!\}_s \{\!|\vartheta_0 \bullet \vartheta_1 \star^T N/y|\!\}_s$;

**Red Suc** $case\ \vartheta_0 \bullet \vartheta_1 suc(M)\ of\ 0 : P\ suc(x) : Q > Q\{\!|\vartheta_0 \bullet \vartheta_1 \star^T M/x|\!\}_s$;

**Red Decrypt** $case\ \vartheta_0 \bullet \vartheta_1 \{M_1, \ldots, M_k\}_N\ of\ \{y_1, \ldots, y_k\}_{\vartheta_0 \bullet \vartheta_1 \star^T N}\ in\ P >$
$P\{\!|\vartheta_0 \bullet \vartheta_1 \star^T M_1/y_1|\!\}_s \ldots \{\!|\vartheta_0 \bullet \vartheta_1 \star^T M_k/y_k|\!\}_s$.

When a process decomposes a term, the involved sub-terms are updated. The intuition for the decryption rule is that we can decrypt a message $\{M\}_N$ only if we use the key $K = \vartheta_0 \bullet \vartheta_1 \star^T N$ which is exactly how the frozen key $N$ should appear to the receiver of the encrypted message. When $K$ and $N$, although starting from different sites, do refer to the same key, the semantic rules decrypt the message and update its relative address by composing $\vartheta_0 \bullet \vartheta_1$ and $M$. The process $P$ then behaves as $P\{\!|\vartheta_0 \bullet \vartheta_1 \star^T M/y|\!\}_s$.

**Localized Commitment Relation.** Eventually, we present in Tab. 4 the extended commitments rules. The *localized commitment relation* is written $P \xrightarrow{\alpha @ \vartheta} A$, where $P$ is a process, $\alpha$ is the action, performed by the sub-process at $\vartheta$, and $A$ is an agent.

The component $\vartheta$ of labels is needed for checking some side conditions of a few rules. Communication rules are successful only if the complementary actions refer to the same name; while for the restriction rule it is necessary to check that there are no clashes between the action and the restricted name. The semantic rules update the messages with the relative addresses of the sender with respect to the receiver. Indeed, the congruence rules lift relative addresses and restrictions as much as needed. For instance, by applying $(\nu r)P_1|P_2 \equiv (\nu \bullet||_0 \star r)(P_1|P_2)$, we have that the restricted name $r$ in $P_1$ appears as $\bullet||_0 \star r$ in the process $(P_1|P_2)$; similarly, by applying the congruence rule $P \mid (\nu r)\langle M\rangle P' \equiv (\nu \bullet||_1 r)\langle \bullet||_1 \star^T M\rangle(P \mid P')$; the term $M$ in $P'$ appears as $\bullet||_1 \star^T M$ in $(P \mid P')$. Finally, also interactions have to update relative addresses.

**Definition 7.4** *Let $F = (x)P_0$ be an abstraction and $C = (\nu\tilde{r})\langle M\rangle P_1$ be a concretion. Then, their* localized interactions *are*

$$F \hat{@} C = (\nu \bullet||_1 \star \tilde{r})(P_0\{\!|\ ||_0 \bullet ||_1 \star^T M/x|\!\}_s \mid P_1)$$
$$C \hat{@} F = (\nu \bullet||_0 \star \tilde{r})(P_1 \mid P_0\{\!|\ ||_1 \bullet ||_0 \star^T M/x|\!\}_s).$$

To see how the localized interactions work, consider $F \hat{@} C$. The restricted names, known as $\tilde{r}$ in $P_1$, are duly updated to $\bullet||_0 \star \tilde{r}$ in the parallel composition of $P_0'$ (i.e. $P_0$ after the substitution) and $P_1$. As for the message, it appears as $M$ in $P_1$ and has to be exported at $P_0$: the term address composition $\star^T$ is therefore applied to the relative address $||_1 \bullet ||_0$ and to $M$. As an example, consider again the processes in Fig. 1 and suppose that $P_0$ is willing to send $\bullet N$ to the process $P_3$. Then, the message will appear as $\bullet||_0||_0 N$ in $(P_0|P_1)$. It will replace, through the routed substitution, the variable $x$ in $(P_2|(P_3|P_4)$ as $||_1 \bullet ||_0||_0 N$. Note that it will arrive to $P_3$ as $||_1||_1||_0 \bullet ||_0||_0 N$, i.e. enriched with the relative address of $P_0$ w.r.t. $P_3$.

## 8 Message Authentication

We can now intuitively present our authentication primitive $[M \stackrel{@}{=} N]$, akin to the matching operator. This "address matching" is passed only if the relative addresses of the two localized terms $M$ and $N$ coincide.

*Comm Out*

$$\frac{}{\overline{r}\langle M\rangle.P \xrightarrow{\overline{r}} (\nu)\langle M\rangle P}$$

*Comm In*

$$\frac{}{r(x).P \xrightarrow{r} (x)P}$$

*Comm Par 1*

$$\frac{P_0 \xrightarrow{\alpha @ \vartheta} A_0}{P_0|P_1 \xrightarrow{\alpha @ ||_0 \vartheta} A_0|P_1}$$

*Comm Par 2*

$$\frac{P_1 \xrightarrow{\alpha @ \vartheta} A_1}{P_0|P_1 \xrightarrow{\alpha @ ||_1 \vartheta} P_0|A_1}$$

*Comm Inter 1*

$$\frac{P_0 \xrightarrow{r' @ \vartheta} F_0 \quad P_1 \xrightarrow{\overline{r} @ \vartheta'} C_1}{P_0|P_1 \xrightarrow{\tau} F_0 \hat{@} C_1} \quad \text{if } r' = ||_0 \vartheta \bullet ||_1 \vartheta' \star r$$

*Comm Inter 2*

$$\frac{P_0 \xrightarrow{\overline{r} @ \vartheta} C_0 \quad P_1 \xrightarrow{r' @ \vartheta'} F_1}{P_0|P_1 \xrightarrow{\tau} C_0 \hat{@} F_1} \quad \text{if } r' = ||_1 \vartheta' \bullet ||_0 \vartheta \star r$$

*Comm Res*

$$\frac{P \xrightarrow{\alpha @ \vartheta} A}{(\nu u)P \xrightarrow{\alpha @ \vartheta} (\nu u)A} \quad \text{if } \epsilon \bullet \vartheta \star \alpha \notin \{u, \overline{u}\}$$

*Comm Red*

$$\frac{P > P' \quad P' \xrightarrow{\alpha @ \vartheta} A'}{P \xrightarrow{\alpha @ \vartheta} A'}$$

*Comm Struct*

$$\frac{P \equiv Q \quad Q \xrightarrow{\alpha @ \vartheta} A \quad A \equiv A'}{P \xrightarrow{\alpha @ \vartheta} A'}$$

Table 4: The localized commitment relation.

The intuition is that if we know which process packed $N$, say $P$, we can also say that $M$ comes indeed from $P$, thus authenticating it. More formally, the extensions due to the new primitive consist in a new case for processes and a new reduction rule.

**Definition 8.1** *Let $M, N$ be two terms as in Def. 6.1, and $l, l' \in \mathrm{Loc}$ be two relative addresses. Then*
$[lM \overset{@}{=} l'N]P$
*is a process, on which we define the following reduction rule:*

**Red Address Match** $[lM \overset{@}{=} lN]P > P$.

Note that free names are prefixed with the empty relative address.

Hereafter, we assume an initial start-up phase, in which processes exchange a message and fix their relative addresses. This can be obtained, e.g., through a preliminary communication between the partners, from $A$ to $B$ on a restricted shared channel. This start-up phase is indeed an abstraction of the preliminary secure exchange of secret information (e.g., long term keys) which is necessary in every cryptographic protocol. We will see an example of this in the next session. This initialization step can be avoided by using our partner authentication primitive; however, for the sake of presentation, we do not combine here the two primitives.

Consider the following simple example, where the process $B$ wants to authenticate a message from $A$ even in the presence of an intruder $E$. The protocol $P$ is:

$$\begin{array}{rcl} P & = & A \mid (B \mid E) \\ A & = & (\nu_\bullet M)\overline{c}\langle \bullet M\rangle.A' \\ B & = & c(x).[x \overset{@}{=} ||_1||_0\bullet||_0 0]B' \end{array}$$

(Recall that $M$ is a name that appears as $\bullet M$ in $A$, by assumption; analogously for $N$ in $E$, below.) Now we show the role that localized names play, and how they guarantee by construction that $B'$ is executed only if the message bound to $x$ has been originated by $A$, i.e., if the message received on channel $c$ is indeed $M$. As said above, the relative address $||_1||_0\bullet||_0$ pointing from $B$ to $A$, encodes the "site" hosting $A$, thus it gives the identity of the process from which $B$ is expecting to receive a message on channel $c$. In order to analyze the behaviour of the protocol in a hostile environment, we consider a generic intruder $E$, as powerful as possible.

We now examine the following two possible message exchanges:

$$\begin{array}{rcl} \text{Message } 1 & : & A \rightarrow B : M \\ \text{Message } 2 & : & E \rightarrow B : N \end{array}$$

The first message represents the correct exchange of message $M$ from $A$ to $B$. The second one is an attempt of $E$ to send a different message $N$ to $B$. The intruder $E$ could actually be of the following form:

$$E = (\nu_\bullet N)\overline{c}\langle \bullet N\rangle.E' + \langle \text{other bad actions}\rangle.$$

The names $M$ and $N$ are received by $B$ prefixed by the relative address corresponding to the respective originators.

Fig. 3 shows the two message exchanges. In particular we see that $M$ is received by $B$ as $||_1||_0\bullet||_0 M$ while $N$ becomes $||_0\bullet||_1 N$. It is now immediate to see that only in the first case $B$ will evolve to $B\{\!|M/x|\!\}_s$,

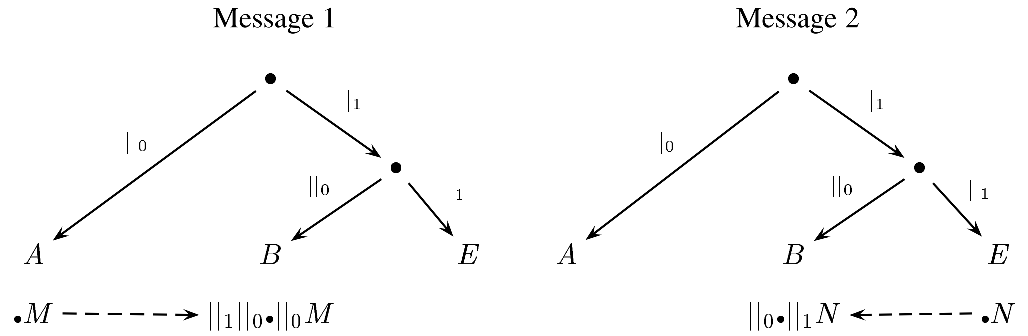


Figure 3: The process $B$ detects that the message $||_1||_0\bullet||_0 M$ is authentic while the message $||_0\bullet||_1 N$ comes from the intruder $E$.

while in the latter it will stop. This is so because only $||_1||_0\bullet||_0 M$ matches with the address of $A$. Every attempt of the intruder of introducing new messages on $c$ is filtered out by the authentication primitive.

A further interesting case arises when the intruder intercepts $M$ and forwards it to $B$. We will show that our mechanism accepts the message as authentic. In particular, we reconsider the previous protocol and we analyze the case of a different intruder that, masquerading as $B$ (written $E(B)$), intercepts $M$ and forwards it to $B$:

$$\begin{array}{lcl} \text{Message 3} & : & A \to E(B) : M \\ \text{Message 4} & : & E \to B : M \end{array}$$

Since $E$ does not, actually *cannot* modify $M$ we would like to accept the message in $B$ even if it has been forwarded by $E$. No matter how many times a message is forwarded, address composition maintains its integrity and the identity of its generator. In detail, $E(B)$ receives $M$ as $||_1||_1\bullet||_0 M$. When $E$ forwards it to $B$, the message is composed with the address of $E$ relative to $B$, $||_0\bullet||_1$, yielding $(||_0\bullet||_1 \star ||_1||_1\bullet||_0)M$. By applying the rule (1) of Def. 3.2 we see that the message is received by $B$ as $||_1||_0\bullet||_0 M$, and therefore it is accepted as authentic. Note also that $B$ can use $||_1||_1\bullet||_0 M$ as a component of a new message $M'$. The receiver $R$ of $M'$ will get $M$ prefixed by the relative address of $A$, say $\vartheta$. So $R$ can check that $M'$ has been packed by $B'$, hence the authenticity of $M'$, and even of its components. Indeed the composition of $\vartheta$ and $||_1||_1\bullet||_0$, via $\star$, gives the address of the originator of $M$ (i.e. $A$), relative to $R$.

We end this section with the following property, in which $\mathcal{C}[-]$ and $\mathcal{C}[-,-]$ are contexts with one and two holes, respectively, and $P\{\!|\tilde{M_i}/\tilde{x_i}|\!\}$ stands for $P\{\!|\tilde{M_1}/\tilde{x_1}|\!\}\ldots\{\!|\tilde{M_n}/\tilde{x_n}|\!\}$.

**Theorem 8.2 (address matching)** *Let $\hat{Q} = \mathcal{C}[R,S]$, $R = c(x).\mathcal{C}_0[[x \overset{@}{=} lN]P_0]$ and $S = \overline{c}\langle M\rangle.P_1$, where the input on c binds the variable $x$ in the matching. Suppose that*
$\hat{Q} > Q \xrightarrow{\tau} Q' = \mathcal{C}'[(\mathcal{C}_0[[x \overset{@}{=} lN]P_0])\{\!|l'M/x|\!\}, P_1] \longrightarrow^* Q'' = \mathcal{C}''[(\mathcal{C}''_0[[x \overset{@}{=} lN]P_0])\{\!|l'M/x|\!\}\{\!|\tilde{M}_i/\tilde{x}_i|\!\}, P'_1]$,
*where $Q@\vartheta\vartheta_0 = R$, $Q@\vartheta\vartheta_1 = S$ and $\mathcal{C}_0[[x \overset{@}{=} lN]P_0]@\vartheta_2 = [x \overset{@}{=} lN]P_0$, for some $\mathcal{C}', \mathcal{C}''_0, \tilde{M}_i, \tilde{x}_i$ and $P'_1$, such that: $Q'@\vartheta\vartheta_0 = (\mathcal{C}_0[[x \overset{@}{=} lN]P_0])\{\!|l'M/x|\!\}$, $Q'@\vartheta\vartheta_1 = P_1$, $Q''@\vartheta\vartheta_1 = P'_1$ and*
$Q''@\vartheta\vartheta_0 = (\mathcal{C}''_0[[x \overset{@}{=} lN]P_0])\{\!|l'M/x|\!\}\{\!|\tilde{M}_i/\tilde{x}_i|\!\}$.
*Then $Q'' > \mathcal{C}''[(\mathcal{C}''_0[P_0]\{\!|l'M/x|\!\})\{\!|\tilde{M}_i/\tilde{x}_i|\!\}, P'_1]$ if and only if $l' = \vartheta_0 \bullet \vartheta_1$ and $l = l' \star \vartheta_2 \bullet \epsilon$.*

PROOF. In the first communication the variable $x$ occurring in the matching is instantiated to $l'' = l' \star \vartheta_2 \bullet \epsilon$, because the input binds $x$ (recall that, in Def. 7.3, the substituting term is enriched while going down in the tree of sequential processes). The sequence of steps leading to $Q''$ only change the contexts $\mathcal{C}', \mathcal{C}_0$ and the process $P_1$, and it possibly binds some variables $\tilde{x_i}$ in $P_0\{\!|l'M/x|\!\}$. Now the reduction on matching can be performed if and only if $[l''M \stackrel{@}{=} lN]$ is at top-level in $\mathcal{C}''$, and $l'' = l' \star \vartheta_2 \bullet \epsilon$ is equal to $l$. ■

# 9 Implementing Authentication

In this section we show that our notion of message authentication based on locations helps studying and analysing cryptographic protocols. The main idea is to observe if a specific authentication protocol is indeed a good "implementation" of our authentication primitive, i.e., if the cryptographic protocol is as strong in detecting names with an "incorrect" relative address as our authentication primitive is.

Recall that in the spi-calculus a compound term, such as an encryption $M$, is considered localized, i.e. its relative address, say $\vartheta_0 \bullet \vartheta_1$, will always point to the process $P$ that made the encryption. In this way, when decrypting $\{M\}_K$ the semantic rules recover the correct addresses for $M$ and $K$ by simply composing the actual address of $\{M\}_K$, $\vartheta_0 \bullet \vartheta_0$, with the (frozen) relative addresses of $M$ and $K$, respectively.

We now show an example of a correct run of the Wide Mouthed Frog key exchange protocol. Consider its simplified version analyzed in [3]. The two processes $A$ and $B$ share keys $K_{AS}$ and $K_{BS}$ with a trusted server $S$. In order to establish a secure channel with $B$, $A$ sends a fresh key $K_{AB}$ encrypted with $K_{AS}$ to the server $S$. Then, the server decrypts the key and forwards it to $B$, this time encrypted with $K_{BS}$. Now $B$ has the key $K_{AB}$ and $A$ can send a message $M$ encrypted with $K_{AB}$ to $B$. The protocol should guarantee that when $B$ receives $M$, such a message has been indeed originated by $A$.

The protocol is composed of the following three messages:

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

Its specification in our calculus with localized names (having mechanically replaced restricted names with their localized counterparts) is:

$$\begin{array}{rcl} P & = & (\nu_\bullet K_{AS})(\nu_\bullet K_{BS})(\ (A|B) \mid S) \\ A & = & (\nu_\bullet K_{AB})(\nu_\bullet M)(\overline{c_{AS}}\langle\{\bullet K_{AB}\}_{||_0||_0 \bullet K_{AS}}\rangle.\overline{c_{AB}}\langle\{\bullet M\}_{\bullet K_{AB}}\rangle) \\ S & = & c_{AS}(x).\mathit{case}\ x\ \mathit{of}\ \{y\}_{||_1 \bullet K_{AS}}\ \mathit{in}\ \overline{c_{BS}}\langle\{y\}_{||_1 \bullet K_{BS}}\rangle \\ B & = & c_{BS}(x).\mathit{case}\ x\ \mathit{of}\ \{y\}_{||_0||_1 \bullet K_{BS}}\ \mathit{in} \\ & & c_{AB}(z).\mathit{case}\ z\ \mathit{of}\ \{r\}_y\ \mathit{in}\ \hat{B} \end{array}$$

Not surprisingly, the specification is in the style of [3], except for localized restricted names. Fig. 4 shows how the localized names are handled in a correct execution of the protocol $P$. Note that $K_{AS}$ and $K_{BS}$ assume a different relative address in the different processes.
After the reception of message 2, $B$ decrypts the received message

$$||_0||_1 \bullet ||_1 \{||_1 \bullet ||_0||_0 K_{AB}\}_{||_1 \bullet K_{BS}}$$

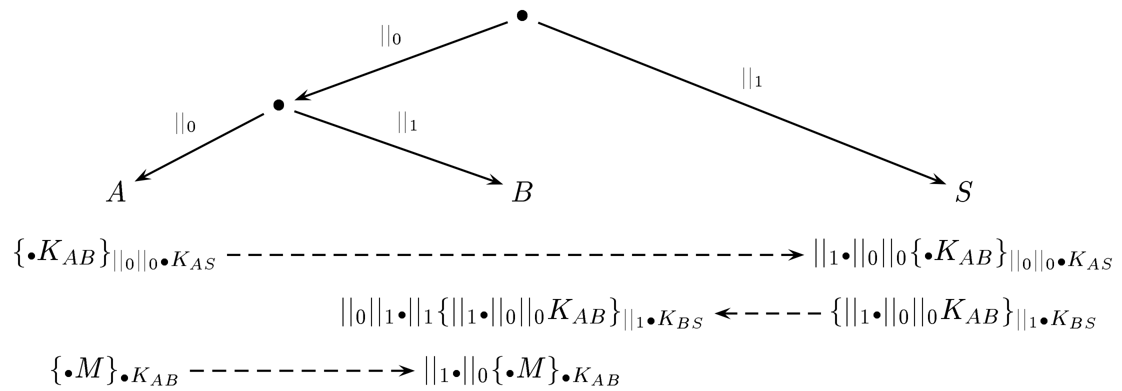


Figure 4: A correct execution of the Wide Mouthed Frog protocol

This means computing

$$\textit{case}\ ||_0||_1\bullet||_1\{||_1\bullet||_0||_0K_{AB}\}_{||_1\bullet K_{BS}}\ \textit{of}\ \{y\}_{||_0||_1\bullet K_{BS}}\ \textit{in}\ c_{AB}(z).$$
$$\textit{case}\ z\ \textit{of}\ \{r\}_y\ \textit{in}\ \hat{B}.$$

The reduction rule **Decrypt** applies because $||_0||_1\bullet K_{BS} = ||_0||_1\bullet||_1 \star^T ||_1\bullet K_{BS}$. The variable $y$ is then set to $(||_0||_1\bullet||_1 \star ||_1\bullet||_0||_0)K_{AB}$, that results in $||_1\bullet||_0K_{AB}$, by the rule $(3)$ of Def. 3.2. This is indeed the correct reference to the key $K_{AB}$ generated by $A$ and installed in its local environment. In the last message $B$ receives $||_1\bullet||_0\{\bullet M\}_{\bullet K_{AB}}$, and succeeds in decrypting it with $||_1\bullet||_0K_{AB}$, obtaining $||_1\bullet||_0M$. The addresses of $M$ and of $A$ relative to $B$ are equal, so $M$ is indeed authentic. This characterizes a "correct" execution.

It is well-known that an attack can take place over two sessions of the protocol above. Basically, it occurs when the intruder replays some messages of the first session in the second one. We follow below the formalization of [3], where this attack is analyzed. Note that in the single session illustrated above no problem arises instead, even if an intruder intercepts the message sent by $A$, then forwarded to $B$. Indeed the message received is the right one, as we have just seen above, as no one can alter neither the relative addresses, nor the encrypted message $\{M\}_{K_{AB}}$.

Now, we show that the attack above is immediately detected by an observer that can compare localized names, e.g. by using our authentication primitive $[M \overset{@}{=} N]$. For the sake of readability, we call $A'$ and $B'$ the two instances of $A$ and $B$ in the second session where $A'$ is trying to send the message $M'$ to $B'$ using the session key $K'_{AB}$:

$$\begin{array}{lrcl}
\text{Message 1} & A \to S & : & \{K_{AB}\}_{K_{AS}} \\
\text{Message 2} & S \to B & : & \{K_{AB}\}_{K_{BS}} \\
\multicolumn{4}{c}{\langle E \text{ eavesdrops Message 2}\rangle} \\
\text{Message 3} & A \to B & : & \{M\}_{K_{AB}} \\
\multicolumn{4}{c}{\langle E \text{ eavesdrops Message 3}\rangle} \\
\text{Message 1}' & A' \to S & : & \{K'_{AB}\}_{K_{AS}} \\
\text{Message 2}' & E(S) \to B' & : & \{K_{AB}\}_{K_{BS}} \\
\text{Message 3}' & E(A') \to B' & : & \{M\}_{K_{AB}}
\end{array}$$

The intruder eavesdrops the first session and then replays messages 2 and 3 in the second session (messages

$2'$ and $3'$). The result is that $B'$ receives a copy of $M$ instead of one of $M'$.

In order to model two parallel session of the protocol, we consider the following specification:

$$P' \;=\; (\nu \bullet K_{AS})(\nu \bullet K_{BS})(\;(\;(A \mid A) \mid (B \mid B)\;) \mid S\;) \mid E$$

where the addresses of localized names $\bullet K_{AS}$ and $\bullet K_{BS}$ are suitably updated in the processes $A$, $B$ and $S$. Note that $A$ generates both $M$ and $K_{AB}$ as fresh names. So each $A$ of $A \mid A$ originates two different messages, say $M$ and $M'$, and two different keys, say $K_{AB}$ and $K'_{AB}$. We also modify $S$ so that it can serve more sessions (for the sake of simplicity we define a server which is just able to handle two sequential sessions):

$$\begin{array}{rcl} S & = & c_{AS}(x).\mathit{case}\ x\ \mathit{of}\ \{y\}_{||_1 \bullet K_{AS}}\ \mathit{in}\ \overline{c_{BS}}\langle \{y\}_{||_1 \bullet K_{BS}} \rangle. \\ & & c_{AS}(z).\mathit{case}\ z\ \mathit{of}\ \{w\}_{||_1 \bullet K_{AS}}\ \mathit{in}\ \overline{c_{BS}}\langle \{w\}_{||_1 \bullet K_{BS}} \rangle \end{array}$$

and we specify the intruder as:

$$E \;=\; c_{BS}(x).c_{AB}(y).\overline{c_{BS}}\langle x \rangle.\overline{c_{AB}}\langle y \rangle,$$

where the names $c_{AB}$, $c_{AS}$ and $c_{BS}$ are free, and thus known to all the processes of $P'$ (unlike $\bullet K_{AS}$ and $\bullet K_{BS}$ that are bound).

Now we can observe the attack sequence in Fig. 5. In particular, when the process $B'$ decrypts message $||_1||_1\bullet||_0||_0\{\bullet M\}_{\bullet K_{AB}}$, it obtains $||_1||_1\bullet||_0||_0 M$ which is a message originated from $A$ and not from $A'$, as it should be. Indeed, the address of $A'$ relative to $B'$ is $||_1||_1\bullet||_0||_1$, and the attack is detected.

We now want to show how the protocol can be done secure by construction through our authentication primitive. The idea is that the last message should be accepted only if it has been originated by the correct initiator. In order to do this we need at least a message from the initiator whose address can be compared with the last message of the protocol. The trick is to add a startup message that securely hooks one initiator with one responder, by sending a fresh message on a restricted channel. The resulting specification follows (the modifications are in bold font and the restricted names are not localized, for the sake of readability):

$$\begin{array}{rcl} P'' & = & (\nu K_{AS})(\nu K_{BS})(\;(\nu \mathbf{startup})((A''|A'') \mid (B''|B'')) \mid S\;) \mid E \\ A'' & = & (\nu K_{AB})(\nu M)(\nu \mathbf{M_A})(\overline{\mathbf{startup}}\langle \mathbf{M_A} \rangle.\overline{c_{AS}}\langle \{K_{AB}\}_{K_{AS}} \rangle.\overline{c_{AB}}\langle \{M\}_{K_{AB}} \rangle) \\ S & = & c_{AS}(x).\mathit{case}\ x\ \mathit{of}\ \{y\}_{K_{AS}}\ \mathit{in}\ \overline{c_{BS}}\langle \{y\}_{K_{BS}} \rangle. \\ & & c_{AS}(z).\mathit{case}\ z\ \mathit{of}\ \{w\}_{K_{AS}}\ \mathit{in}\ \overline{c_{BS}}\langle \{w\}_{K_{BS}} \rangle \\ B'' & = & \mathbf{startup}(\mathbf{m_a}).c_{BS}(x).\mathit{case}\ x\ \mathit{of}\ \{y\}_{K_{BS}}\ \mathit{in} \\ & & c_{AB}(z).\mathit{case}\ z\ \mathit{of}\ \{r\}_y\ \mathit{in}\ [\mathbf{r} \overset{@}{=} \mathbf{m_a}]\hat{B} \end{array}$$

In $P''$ the two $B''$ processes receive by the two $A''$ two different startup messages, with two different addresses. It is thus no longer possible for the intruder to carry out a replay attack. In fact, the cheated $B''$ will be able to stop before delivering the message to $\hat{B}$. By comparing the traces of this protocol correct "by construction" with the traces of the previous one it is easy to see that they are not equivalent. A potential attack is thus detected.

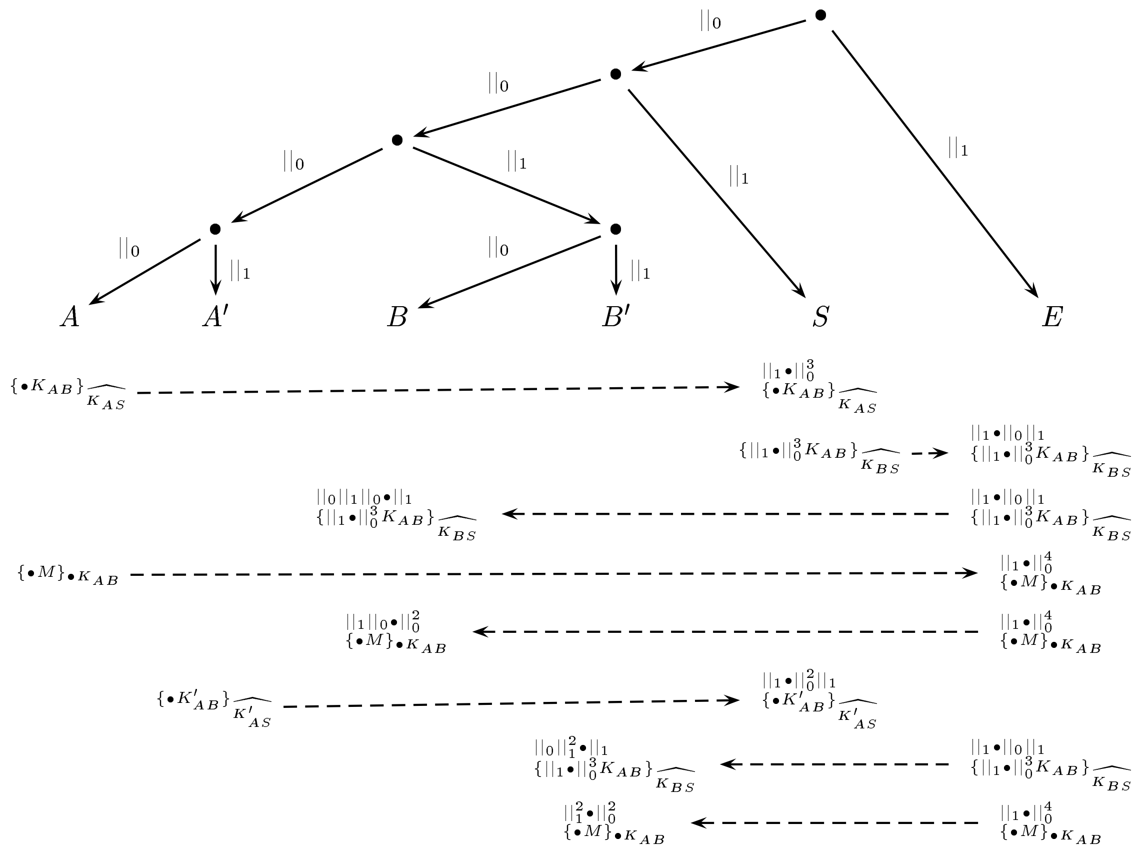

Figure 5: An attack on the Wide Mouthed Frog protocol; where $||_i^k$ stands for a sequence of $k$ tags $||_i$, and where $\widehat{K_{AS}} = ||_0^3 \bullet K_{AS}$, $\widehat{K_{BS}} = ||_1 \bullet K_{BS}$ and $\widehat{K'_{AS}} = ||_0^2 ||_1 \bullet K_{AS}$.

## 10 Conclusions and Future Work

We defined two primitives that guarantee partner and message authentication over public channels, based on the same semantic feature derived form the proved transition systems [9]. Partner authentication is based on a semantics of the $\pi$-calculus where names of channels are indexed with the expected relative addresses of the communicating parties. In particular, any time two processes try to communicate over a common channel, their relative addresses are checked against the index of the channel used. The communication is enabled only if the check is passed, i.e. if the relative addresses are compatible with the indexes. Moreover, the very same channel can be used in a multiplexing fashion: two processes, say $P$ and $Q$, can go on exchanging messages on channel $c$, interleaving their activity with that of another pair of processes, say $R$ and $S$, using $c$ as well. It will be never the case that a message for $P$ is read by $R$ or comes from $S$, unless this is indeed the intended behaviour of both $P$ and $R$ or both $P$ and $S$.

Message authentication is based on a semantics of the spi-calculus where each message $M$ is localized, via a relative address $l$, to the process $P$ that packed $M$. The authentication primitive compares the relative address $l$ with the address $l'$ of a process $Q$, relative to the receiver of $M$. The check succeeds and authenticates the message $M$ only if $l = l'$, expressing that it was indeed the process $Q$ that packed $M$.

Our two primitives are in a sense orthogonal, as they operate on independent features of the calculi considered. Of course, one may combine them, e.g. by carrying on the spi-calculus the notion of located channel introduced for the $\pi$-calculus. Both notions of authentication can then be guaranteed "by construction".

Note that our partner authentication primitive does not transform a public channel into a private one. Indeed, partner authentication clearly separates the concepts of authentication and secrecy. More importantly, once two processes communicating on public channels are hooked it is impossible for a third process to interfere in the communication.

Our notion of message authentication does not need private channels, as well. A message $M$ may be considered authentic even if it has been intercepted or eavesdropped, i.e., our mechanism does not guarantee the secrecy of $M$, but only that $M$ has been generated or packed by the claimed entity. Thus, our primitive corresponds neither to a private channel in the basic $\pi$-calculus, nor to a cryptographic one in the spi-calculus: both appear to be too strong to message authentication alone, as they guarantee also secrecy.

The idea of exploiting locations for the analysis of authentication comes from [14], where however entities are bound to physical addresses of the net. An approach related is Abadi-Fournet-Gonthier's [2], in which principals have explicit, fixed names (see also the Join-calculus [19] and SEAL [33]). Here we relaxed the rigidity of a fixed mapping of sites, by introducing a sort of "identifiers of sites" represented by relative addresses. As a matter of fact, the actual placement of a process on a site can be recovered by composing our localized names (akin to the environment function of sequential languages) with allocation tables (similar to a store). Actually, [14] models a wider notion of authentication, that we plan to investigate next.

As discussed in the paper, our primitive may be not implementable directly. Indeed, one should have a low-level, highly reliable mechanism to manage localized names, which is unrealistic in many cases, but possible for instance in LAN or virtual private networks. A further step could be encrypting relative addresses within the header of messages, in the style of IPsec [31]. Nevertheless, our proposal can help reasoning on authentication and security from an abstract point of view. This is indeed the main aim of our approach and we are presently developing some ideas that we briefly describe in the following.

First, it could be possible to verify the correctness of a cryptographic protocol by showing that its messages implement partner authentication when needed. As an example, a typical challenge-response technique requires to send a nonce (random challenge) and to expect it back, encrypted with a secret shared key. Challenge-response can be proved to implement our located input actions, under some suitable conditions. The proofs that implementations satisfy specifications are often hard, just because private channels are used to model authenticated channels. Indeed, private channels often seem too far from cryptographic implementations. So our proposal can help, as we need no private channels.

Moreover, we could verify if a cryptography-based protocol ensures message authentication, by checking a version of it containing also the primitive $[M \overset{@}{=} N]$: the original formulation and ours should exhibit the same behaviour. This check of specifications against implementations, is much in the style of the congruence-based techniques typical of process calculi (see, e.g., [3]).

Finally, we feel confident that our proposal scales up, because some languages for concurrent and reactive systems, like Facile [32], PICT [28], CML [26], Esterel [4] are built on top of a core process calculus like the one we use here; also, they have an operational semantics that can easily be turned into a proved one, as the successful cases of Facile [11] and Esterel [23] show. Of course a great deal of work is still necessary to make our proposal applicable in real cases.

## References


[1] M. Abadi. 'Secrecy by Typing In Security protocols". *Journal of the ACM*, 5(46):18–36, sept 1999. An abstract appeared in *Proceedings of TACS 1997*. Springer-Verlag, 1997.

[2] M. Abadi, C. Fournet, and G. Gonthier. "Secure Implementation of Channel Abstractions". In *Proceedings of the* $13^{th}$ *Annual IEEE Symposium on Logic in Computer Science*, pages 105–116, 1998.

[3] M. Abadi and A. D. Gordon. "A Calculus for Cryptographic Protocols: The Spi Calculus". *Information and Computation 148*, 1:1–70, January 1999.

[4] G. Berry and L. Cosserat. "The Synchronous Programming Language ESTEREL and its Mathematical Semantics". In *Seminar on Concurrency, LNCS 197*. Springer-Verlag, 1984.

[5] C. Bodei. *Security Issues in Process Calculi.* PhD thesis, Dipartimento di Informatica, Università di Pisa. TD-2/00, March, 2000.

[6] C. Bodei, P. Degano, and C. Priami. "Names of the $\pi$-Calculus Agents Handled Locally". *Theoretical Computer Science*, 253(2):155–184, February 2000. Extended abstract in: *Proceedings of ICALP'96*, LNCS 1099, Springer-Verlag.

[7] M. Burrows, M. Abadi, and R. Needham. "A Logic of Authentication". *ACM Transactions on Computer Systems*, pages 18–36, February 1990.

[8] C. Bodei, P. Degano, R. Focardi, and C. Priami. "Authentication via Localized Names". In *Proceedings of the 12th Computer Security Foundation Workshop*, pages 98–110. IEEE press, 1999.

[9] P. Degano and C. Priami. "Enhanced Operational Semantics: A Tool for Describing and Analysing Concurrent Systems". To appear in ACM Computing Surveys.

[10] P. Degano and C. Priami. "Non Interleaving Semantics for Mobile Processes". *Theoretical Computer Science*, 216:237–270, 1999.

[11] P. Degano, C. Priami, L. Leth, and B. Thomsen. "Causality for debugging mobile agents". *Acta Informatica*, 1999.

[12] A. Durante, R. Focardi, and R. Gorrieri. "A Compiler for Analysing Cryptographic Protocols Using Non-Interference". To appear on ACM TOSEM.

[13] F. J. T. Fabrega, J. C. Herzog, and J. D. Guttman. "Strand Spaces: Why is a Security Protocol Correct?". In *Proceedings of the 1998 Symposium on Security and Privacy*. IEEE Computer Society Press, May 1998.

[14] R. Focardi. "Using Entity Locations for the Analysis of Authentication Protocols". In *Proceedings of Sixth Italian Conference on Theoretical Computer Science*, November 1998.

[15] R. Focardi, A. Ghelli, and R. Gorrieri. "Using Non Interference for the Analysis of Security Protocols ". In *Proceeding of the DIMACS Workshop on Design and Formal Verification of Security Protocols (H. Orman and C. Meadows Ed.)*, DIMACS Center, Rutgers University, September 1997.

[16] R. Focardi, R. Gorrieri, and F. Martinelli. "Non Interference for the Analysis of Cryptographic Protocols". In *Proceedings of ICALP'00*, Geneva (Switzerland), July 2000.

[17] R. Focardi and F. Martinelli. "A Uniform Approach for the Definition of Security Properties". In *Proceedings of World Congress on Formal Methods in the Development of Computing Systems, LNCS 1708*, pages 794–813, Toulouse (France), September 1999. Springer-Verlag LNCS.

[18] International Organization for Standardization. Information technology - Security techniques - Entity authentication mechanism; Part 1: General model. ISO/IEC 9798-1, Second Edition, September 1991.

[19] C. Fournet and Gonthier. The reflexive chemical abstract machine and the join calculus. In *Proceedings of the* $23^{rd}$ *ACM Symposium on Principles of Programming Languages*, 1996.

[20] R. Kemmerer, C. Meadows, and J. Millen. "Three systems for cryptographic protocol analysis". *J. Cryptology*, 7(2):79–130, 1994.

[21] G. Lowe. "Breaking and Fixing the Needham-Schroeder Public-key Protocol using FDR". In *Proceedings of Second International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)*, pages 146–166, Passau (Germany), March 1996. Springer-Verlag, LNCS 1055.

[22] G. Lowe. "A Hierarchy of Authentication Specification". In *Proceedings of the 10th Computer Security Foundation Workshop*. IEEE press, 1997.

[23] A. Maggiolo-Schettini and S. Tini. "Applying techniques of asynchronous concurrency to synchronous languages". *Fundamenta Informaticae*, 40:221–250, 1999.

[24] R. Milner, J. Parrow, and D. Walker. "A Calculus of Mobile Processes (I and II)". *Information and Computation*, 100(1):1–77, 1992.

[25] J. C. Mitchell, M. Mitchell, and U. Stern. "Automated Analysis of Cryptographic Protocols Using Mur$\phi$". In *Proceedings of the 1997 IEEE Symposium on Research in Security and Privacy*, pages 141–153. IEEE Computer Society Press, 1997.

[26] F. Nielson and H. R. Nielson. “From CML to its Process Algebra”. *Theoretical Computer Science*, 155:179–219, 1996.

[27] National Bureau of Standards. “Data Encryption Standard (DES)”. FIPS Publication 46, 1977.

[28] Benjamin C. Pierce and David N. Turner. PICT: A programming language based on the pi-calculus. In Gordon Plotkin, Colin Stirling, and Mads Tofte, editors, *Proof, Language and Interaction: Essays in Honour of Robin Milner*, Foundations of Computing. mit, May 2000.

[29] S. Schneider. “Verifying authentication protocols in CSP”. *IEEE Transactions on Software Engineering*, 24(9), September 1998.

[30] B. Schneier. *Applied Cryptography*. John Wiley & Sons, Inc., 1996. Second edition.

[31] R. Thayer, N. Doraswamy, and R. Glenn. RFC 2411: IP security document roadmap, November 1998.

[32] B. Thomsen, L. Leth, S. Prasad, T.-M. Kuo, A. Kramer, F. Knabe, and A. Giacalone. “Facile Antigua Release Programming Guide”. Technical Report ECRC-93-20, European Computer-Industry Research Centre, 1993.

[33] J. Vitek and G. Castagna. “Seal: A framework for secure mobile computations”. In *Internet Programming Languages*, 1999.