Hide and New in the π-calculus

Marco Giunti
CITI and DI-FCT, Universidade Nova de Lisboa, Portugal[*]

Catuscia Palamidessi Frank D. Valencia
INRIA Saclay and LIX, Ecole Polytechnique, France [†]

In this paper, we enrich the π-calculus with an operator for confidentiality (*hide*), whose main effect is to restrict the access to the *object of the communication*, thus representing confidentiality in a natural way. The *hide* operator is meant for local communication, and it differs from *new* in that it forbids the extrusion of the name and hence has a static scope. Consequently, a communication channel in the scope of a *hide* can be implemented as a dedicated channel, and it is more secure than one in the scope of a *new*. To emphasize the difference, we introduce a *spy* context that represents a side-channel attack and breaks some of the standard security equations for *new*. To formally reason on the security guarantees provided by the *hide* construct, we introduce an observational theory and establish stronger equivalences by relying on a proof technique based on bisimulation semantics.

1 Introduction

The restriction operator is present in most process calculi. Its behaviour is crucial for *expressiveness* (e.g., for specifying unbounded linked structures, nonce generation and locality). In the π-calculus [19, 20], it plays a prominent role: It provides for the generation and extrusion of unique names. In CCS [18], it is also fundamental but it does not provide for name extrusion: It limits the interface of a given process with its external world. In this paper we shall extend the π-calculus with a hiding operator, called hide, that behaves similarly to the CCS restriction. The motivation for our work comes from the realm of *secrecy* and *confidentiality*: we shall argue that hide allows us to express and guarantee secret communications.

Motivation. Secrecy and confidentiality are major concerns in most systems of communicating agents. Either because some of the agents are untrusted, or because the communication uses insecure channels, there may be the risk of sensitive information being leaked to potentially malicious entities. The price to pay for such security breaches may also be very high. It is not surprising, therefore, that secrecy and confidentiality have become central issues in the formal specification and verification of communicating systems.

The π-calculus and especially its variants enriched with mechanisms to express cryptographic operations, the spi calculus [5] and the applied π-calculus [3], have become popular formalisms for security applications. They all feature the operator new (restriction) and make crucial use of it in the definition of security protocols. The prominent aspects of new are the capability of creating a new channel name, whose use is restricted within a certain scope, and the possibility of enlarging its scope by communicating it to other processes. The latter property is central to the most interesting feature of the π-calculus: the *mobility* of the communication structure.

[*] Work partially supported by the project PTDC/EIA-CCO/117513/2010 Liveness, Statically. The work has been done during the period that the author spent at LIX, Ecole Polytechnique, with the support of an ERCIM postdoc fellowship. The author would like to thank INRIA and ERCIM for such oppurtunity.

[†] Work partially supported by the project ANR-09-BLAN-0169-01 PANDA

B. Luttik and M.A. Reniers (Eds.): Combined Workshop on Expressiveness in
Concurrency and Structural Operational Semantics (EXPRESS/SOS 2012)
EPTCS 89, 2012, pp. 65–79, doi:10.4204/EPTCS.89.6

Although in principle the restriction aspect of new should guarantee that the channel is used for communication within a secure environment only, the capability of extruding the scope leads to security problems. In particular, it makes it unnatural to implement the communication using dedicated channels, and non-dedicated channels are not secure by default. The spi calculus and the applied π-calculus do not assume, indeed, any security guarantee on the channel, and implement security by using cryptographic encryption.

Let us illustrate the problem with an example. The following π-calculus process describes a protocol for the exchange of a confidential information:

$$P = \overline{s}\langle \mathrm{CreditCard}\rangle \mid s(x).\mathsf{if}\, x = \mathrm{OwnerCard}\,\mathsf{then}\,(\overline{p}\langle \mathrm{Ok}\rangle \mid \overline{p}\langle s\rangle) \qquad p \neq s$$

In this specification, the thread on the left sends a credit card number over the channel s to the thread on the right which is waiting for an input on the same channel. If the received card number is the expected one, then the latter both sends an ack and forwards the communication channel s on a public channel p. The problem is that, while the confidentiality of the information would require the context to be unable to interfere with the protocol and to steal the credit card number, in fact this is not guaranteed in the π-calculus where interaction with a parallel process waiting for input on channel s is allowed.

To amend this problem, the idea is to let the channel for the exchange of the secret information available only to the process P, restricting its scope to P with the declaration: $(\mathsf{new}\, s)P$. The π-calculus semantics makes the exchange invisible to the context. This is formalized by the following observational equation stating that no π-calculus context can tell apart P from its continuation:

$$(\mathsf{new}\, s)P \cong^{\mathrm{obs}}_{\pi} (\mathsf{new}\, s)\mathsf{if}\ \mathrm{CreditCard} = \mathrm{OwnerCard}\,\mathsf{then}\,(\overline{p}\langle \mathrm{Ok}\rangle \mid \overline{p}\langle s\rangle) \tag{1}$$

Unfortunately, to preserve such behavioral equations when processes are deployed in untrusted environments is difficult, since, as explained above, we cannot rely on dedicated channels for communication on names created by the new operator. One natural approach to cope with this problem is to map the private communication within the scope of the new into open communications protected by cryptography.

For instance, the process $(\mathsf{new}\, s)P$ could be implemented in the spi calculus protocol $\llbracket(\mathsf{new}\, s)P\rrbracket$ below by using a public-key crypto-scheme. In this implementation the creation of a π-calculus channel s is mapped into the creation of a couple of spi calculus keys: a public key s^+ and a private key s^-. The receiver performs decryption of the crypto-packet $\{\mathrm{CC}\}_{s^+}$ with the private key s^-; the operation assigns the card number to the variable in the conditional test.

$$\begin{aligned} \llbracket(\mathsf{new}\, s)P\rrbracket &\stackrel{\mathrm{def}}{=} (\mathsf{new}\, s^+, s^-)(\overline{net}\langle\{\mathrm{CC}\}_{s^+}\rangle.\mathbf{0} \mid net(y).\mathsf{decrypt}\, y\, \mathsf{as}\, \{x\}_{s^-}\, \mathsf{in}\, Q) \\ Q &\stackrel{\mathrm{def}}{=} \mathsf{if}\, x = \mathrm{OC}\, \mathsf{then}\, \overline{net}\langle\{\mathrm{Ok}\}_{p^+}\rangle \mid \overline{net}\langle\{s^+, s^-\}_{p^+}\rangle \end{aligned}$$

Unfortunately, the naive protocol above suffers from a number of problems, among which the most serious is the lack of forward secrecy [1]: this property would guarantee that if keys are corrupted at some time t then the protocol steps occurred before t do preserve secrecy. In particular, forward secrecy requires that the content of the packet $\{\mathrm{CC}\}_{s^+}$, which is the credit card number, is not disclosed if at some step of the computation the context gains the decryption key s^-. Stated differently, the implementation $\llbracket\cdot\rrbracket$ should preserve the semantics of equation (1): that is, it should be fully abstract. It is easy to see that this is not the case since a spi calculus context can first buffer the encrypted packet and subsequently, whenever it enters in posses of the decryption key, retrieve the confidential information; this breaks equation (1). While a solution to recover the behavioral theory of π-calculus is available [11], the price to pay is a complex cryptographic protocol that relies on a set of trusted authorities acting as proxies.

Based on these considerations, in this paper we argue that the restriction operator of π-calculus does not adequately ensure confidentiality. To tackle this problem, we introduce an operator to program explicitly secret communications, called hide. From a programming language point of view, the envisaged use of the operator is for declaring secret a medium used for *local* inter-process communication; examples include pipelines, message queues and IPC mechanisms of microkernels. The operator is static: that is, we assume that the scope of hidden channels can not be extruded. The motivation is that all processes using a private channel shall be included in the scope of its hide declaration; processes outside the scope represent another location, and must not interfere with the protocol. Since the hide cannot extrude the scope of secret channels, we can use it to directly build specifications that preserves forward secrecy. In contrast, we regard the restriction operator of the π-calculus, new, as useful to create a new channel for message passing with scope extrusion, and which does not provide secrecy guarantees.

To emphasize the difference between hide and new, we introduce a *spy* context that represents a side-channel attack on the non-dedicated channels. In practice, *spy* is able to detect whether there has been a communication on one of the channels not protected by a hide, but is not able to retrieve its content.

Contributions. We introduce the *secret* π-calculus as an extension of the π-calculus with an operator representing confidentiality (hide). We develop its structural operational semantics and its observational theory. In particular, we provide a reduction semantics, a labelled transition semantics and an observational equivalence. We show that the observational equivalence induced by the reduction semantics coincides by the labelled transition system semantics. To illustrate the difference between hide and new, we shall also consider a distinguished process context, called *spy*, representing a side-channel attack.

Plan of the paper In the next section we introduce the syntax and the reduction semantics of the secret π-calculus. In Section 3 we present the observational equivalence, and a characterization based on labelled transition semantics, that we show sound and complete. In Section 4 we introduce the *spy* process, and we extend the reduction semantics and bisimulation method accordingly. In Section 5 we discuss some algebraic equalities and inequalities of the secret π-calculus, and we analyze some interesting examples, notably an implementation of name matching, and a deployment of mandatory access control. Finally, Section 6 presents related work and concludes. An extended version of the paper containing all proofs is available online [15].

2 Secret π-calculus

This section introduces the syntax and the semantics of our calculus, the *secret π-calculus*. The syntax of the processes in Figure 1 extends that of the π-calculus [19, 20] by: (1) We consider two binding operators: new , which – as we will argue – does not offer enough security guarantees, and hide, which serves to program secrecy. (2) We use two forms of restricted pattern matching in input, so that we can deny a process to receive a (possibly empty) set of channels, or we can enforce a process to receive only trusted channels. When in the first form the set of channels is empty we have the standard input of π-calculus. We use an infinite set of names $\mathscr{N}$, ranged over by $a,b,\dots,x,y,z$, to represent channel names and parameters, i.e. the subjects and the objects of communication, respectively. We let A,B range over subsets of $\mathscr{N}$.

A process of the form $x(y \div B).P$ represents an input where the name x is the input channel name, y is a formal parameter which can appear in the continuation P, and B is the set of *blocked* names that the process cannot receive. On contrast, an input process of the form $x[y : A].P$ declares the object names that the process can *accept*: that is, the process accepts in input a name z only if $z \in A$. This permits to program

P,Q ::=		Processes:		
	$x(y \div B).P$	input	$(\mathsf{new}\,x)(P)$	restriction
	$x[y : A].P$	trusted input	$[\mathsf{hide}\,x][P]$	secrecy
	$\overline{x}\langle y\rangle.P$	output	$\mathbf{0}$	inaction
	$P \mid Q$	composition	$!P$	replication

Figure 1: Syntax of the secret π-calculus

security protocols where only trusted names can be received. The free and the bound names of such process are defined as follows: $\mathrm{fn}(x[y \div B].P) = (\mathrm{fn}(P) \setminus \{y\}) \cup \{x\} \cup B$ and $\mathrm{bn}(x[y \div B].P) = \{y\} \cup \mathrm{bn}(P)$, $\mathrm{fn}(x(y : A).P) = (\mathrm{fn}(P) \setminus \{y\}) \cup \{x\} \cup A$ and $\mathrm{bn}(x(y : A).P) = \{y\} \cup \mathrm{bn}(P)$.

Processes $\overline{x}\langle y\rangle.P$, $(\mathsf{new}\,x)(P)$, $P \mid Q$, $!P$, and $\mathbf{0}$ are the pi calculus operators respectively describing an output of a name y over channel x, restriction of x in P, parallel composition, replication and inaction; see [23] for more details.

The process $[\mathsf{hide}\,x][P]$ represents a process P in which the name x is regarded as *secret*, and should not be accessible to any process external to P. $[\mathsf{hide}\,x][P]$ binds the occurrence of x in P: $\mathrm{fn}([\mathsf{hide}\,x][P]) = \mathrm{fn}(P) \setminus \{x\}$, and $\mathrm{bn}([\mathsf{hide}\,x][P]) = \{x\} \cup \mathrm{bn}(P)$.

Contexts are processes containing a hole $-$. We write $C[P]$ for the process obtained by replacing $-$ with P in $C[-]$.

$$C[-] \ ::= \ - \ \mid \ C[-] \mid P \ \mid \ P \mid C[-] \ \mid \ (\mathsf{new}\,x)[-] \ \mid \ [\mathsf{hide}\,x][-] \qquad \text{contexts}$$

We write $x(y).P$ as a short of $x(y \div \emptyset).P$, and omit curly brackets in $x(y \div \{b\}).P$ and $x[y : \{a\}].P$. When no ambiguity is possible, we will remove scope parentheses in $(\mathsf{new}\,x)(P)$ and $[\mathsf{hide}\,x][P]$. We will often avoid to indicate trailing $\mathbf{0}$s.

The combination of the accept and the block construct permits to design processes which are not subject to interference attacks from the context. We note that their role is dual: the accept operator prevents the reception (intrusion) of untrusted names from the environment, and its use is specified by the programmer. The block mechanism prevents another process from sending (extruding) a secret name, and it is inserted automatically by the system to ensure the protection of such names. One may wonder whether we could have used just one form of (trusted) input, and declare the names to be blocked by accepting all names in $\mathcal{N}$ but the intended ones. The main reason that guided our choice is that we believe that our form of input with blocked names can be effectively implemented, for instance by using blacklists. Also, we think that there is a nice symmetry among processes $x(y \div B).P$ and $(\mathsf{new}\,x)P$, and among processes $x[y : A].P$ and $[\mathsf{hide}\,x]P$.

We embed the block mechanism in the rules for structural congruence through the operation $\uplus$ defined in Figure 2. *Blocked* names could indeed be introduced both statically and dynamically, i.e. when structural congruence is performed during the computation. We leave the time when the system blocks explicitly the name in components as an implementation detail. Note that in the second rule of the first line the name b is guaranteed to be different from all the names in A, because in the congruence rule for *hide* (cfr same Figure) the free names of Q are required to be different from the name we want to hide, so the alpha conversion should be applied .

Following standard lines, we define the semantics of our calculus via a reduction relation, also specified in Figure 2. We assume a capture-free substitution operation $\{z/y\}$: the process $P\{z/y\}$ is obtained

Rules for blocking a name

$$(x(y \div B).P) \uplus b \overset{\text{def}}{=} x(y \div B \cup \{b\}).(P \uplus b) \qquad (x[y:A].P) \uplus b \overset{\text{def}}{=} x[y:A].(P \uplus b)$$

$$((\mathsf{new}\,x)(P)) \uplus b \overset{\text{def}}{=} (\mathsf{new}\,x)(P \uplus b)^{*} \qquad ([\mathsf{hide}\,x][P]) \uplus b \overset{\text{def}}{=} [\mathsf{hide}\,x][P \uplus b]^{*} \quad (*)\, b \neq x$$

$$(\overline{x}\langle y \rangle.P) \uplus b \overset{\text{def}}{=} \overline{x}\langle y \rangle.(P \uplus b) \qquad (P \mid Q) \uplus b \overset{\text{def}}{=} P \uplus b \mid Q \uplus b$$

$$(!P) \uplus b \overset{\text{def}}{=} !(P \uplus b) \qquad \mathbf{0} \uplus b \overset{\text{def}}{=} \mathbf{0}$$

Rules for structural congruence

$$P \mid Q \equiv Q \mid P \qquad (P \mid Q) \mid J \equiv P \mid (Q \mid J) \qquad !P \equiv P \mid !P$$

$$(\mathsf{new}\,x)(\mathbf{0}) \equiv \mathbf{0} \qquad [\mathsf{hide}\,x][\mathbf{0}] \equiv \mathbf{0}$$

$$(\mathsf{new}\,x)(P) \mid Q \equiv (\mathsf{new}\,x)(P \mid Q) \quad x \notin \mathrm{fn}(Q)$$

$$[\mathsf{hide}\,x][P] \mid Q \equiv [\mathsf{hide}\,x][P \mid Q \uplus x] \qquad x \notin \mathrm{fn}(Q)$$

$$(\mathsf{new}\,x)([\mathsf{hide}\,y][P]) \equiv [\mathsf{hide}\,y][(\mathsf{new}\,x)(P)] \quad x \neq y$$

Reduction rules

$$\frac{z \notin B}{x(y \div B).P \mid \overline{x}\langle z \rangle.Q \rightarrow P\{z/y\} \mid Q} \qquad \text{[R-COM]}$$

$$\frac{z \in A}{x[y:A].P \mid \overline{x}\langle z \rangle.Q \rightarrow P\{z/y\} \mid Q} \qquad \text{[R-T-COM]}$$

$$\frac{P \rightarrow P'}{(\mathsf{new}\,x)(P) \rightarrow (\mathsf{new}\,x)(P')} \qquad \frac{P \rightarrow P'}{[\mathsf{hide}\,x][P] \rightarrow [\mathsf{hide}\,x][P']} \qquad \text{[R-NEW],[R-HIDE]}$$

$$\frac{P \rightarrow P'}{P \mid Q \rightarrow P' \mid Q} \qquad \frac{P \equiv Q \quad Q \rightarrow Q' \quad Q' \equiv P'}{P \rightarrow P'} \qquad \text{[R-PAR],[R-STRUCT]}$$

Figure 2: Reduction semantics

from P by substituting all the free occurrences of y by z. As usual, we use a structural congruence $\equiv$ to rearrange processes. Such congruence includes the equivalence induced by alpha-conversion, and the relations defined in Figure 2. The rules for the π-calculus operators (first line) are the standard ones. The rules for inaction under a binder follow (second line). We recall that the scope extrusion rule for new (third line) permits to enlarge the scope of a name and let a process receive it. In contrast, the scope extrusion rule for hide (fourth line) permits to enlarge the scope of a name, but at the same time it sets the name to *blocked* for the process which are being included in the scope, thus preventing them to receive the name. The last rule (fifth line) permits to swap the two binders.

The first rule for reduction, [R-COM], says that an input process of the form $x(y \div B).P$ is allowed to synchronize with an output process $\overline{x}\langle z \rangle.Q$ and receive the name z provided that z is not *blocked* ($z \notin B$). The result of the synchronization is the progression of both the receiver and the sender, where the formal parameter in the input's continuation is replaced by the name z. Note that whenever $B = \emptyset$ we have the standard communication rule of the π-calculus. The main novelty is represented by the rule for trusted communication [R-T-COM]. This rule says that an output process can send a name z over x to a parallel process waiting for input on x, provided that z is explicitly declared as accepted ($z \in A$) by the receiver. If this is the case, the name will replace the occurence of the formal parameter in the input's continuation.

Rules [R-NEW] and [R-HIDE] are for new and for hide respectively, and follow the same schema. The rules for parallel composition, replication and incorporating structural congruence are standard.

We let $P \Rightarrow P'$ whenever either (a) $P \to \cdots \to P'$, or (b) $P' = P$.

Example 2.1. *We show how* hide *can be used to prevent the extrusion of a secret. Consider the process:*

$$P \overset{def}{=} [\mathsf{hide}\, z][\overline{x}\langle v\rangle] \qquad x \neq z$$

*The process $\overline{x}\langle v\rangle$ can be interpreted as an internal attacker trying to leak the name v to a context $C[-] \overset{def}{=} - \mid x(y).\overline{leak}\langle y\rangle$. By using the structural rule for enlarging the scope of hide in Figure 2 we infer that $C[P] \equiv [\mathsf{hide}\, z][\overline{x}\langle v\rangle \mid x(y \div z).\overline{leak}\langle y\rangle]$. Whenever the name v is not declared secret, that is whenever $v \neq z$, the leak cannot be prevented: by applying [*R-COM*],[*R-HIDE*], and [*R-STRUCT*] we have $C[P] \to \overline{leak}\langle v\rangle$. Conversely, when the name v is protected by* hide*, that is $v = z$, we do not have any interaction and secrecy is preserved.*

Example 2.2. *The combined use of the accept and block sets permits to avoid interference with the context. Consider the process below, where $n > 0$:*

$$P \overset{def}{=} [\mathsf{hide}\, z_1]\cdots[\mathsf{hide}\, z_n][\cdots[x[y:Z].P \mid \overline{x}\langle z_i\rangle]\cdots] \qquad\qquad Z \subseteq \{z_1, \cdots z_n\}, i \in \{1,\ldots,n\}$$

Take a context $C[-] \overset{def}{=} - \mid (\mathsf{new}\, y)!\overline{x}\langle y\rangle \mid !x(w)$. Such context is unable to send the fresh name y to P, because the input process in P is programmed to accept only trusted names protected by hide*. Dually, the context cannot receive the protected name z_i. Therefore C and P cannot interact: $C[P] \to Q$ implies that a) $Q \equiv C[[\mathsf{hide}\, z_1]\cdots[\mathsf{hide}\, z_n][\cdots[P\{z_i/y\}]\cdots]]$ or b) $Q \equiv C[P]$.*

3 Observational equivalence

In this section we define a notion of behavioral equivalence based on observables, or barbs. As the reader will notice, a distinctive feature of our observational theory is that trusted inputs are visible only under certain conditions, namely that the context knows at least a name that is declared as accepted. Conversely, processes trying to send a name protected by an hide declaration are not visible at all. The choice to work in a synchronous setting permits us to emphasize the differences among our theory and that of π-calculus. However, the same results would hold for a secret asynchronous π-calculus, while the contrast would be less explicit as input barbs would not be observable.

We say that a name x is bound in P if $x \in \mathrm{bn}(P)$. An occurrence of y is hidden in P if such occurrence of y appears in the scope of a hide operator in P.

Definition 3.1 (Barbs)**.** *We define:*

- *$P\downarrow_x$ whenever $P \equiv C[x[y:A].Q]$ with x not bound in P and $A \cap \mathrm{bn}(P) \neq A$, or whenever $P \equiv C[x(y \div B).Q]$ with x not bound in P.*
- *$P\downarrow_{\overline{x}}$ whenever $P \equiv C[\overline{x}\langle y\rangle.Q]$ with x not bound in P and y not hidden in P.*

Based on this definition, we have that $P_1 \overset{\mathrm{def}}{=} [\mathsf{hide}\, x]z[y:x].Q$, $P_2 \overset{\mathrm{def}}{=} (\mathsf{new}\, x)x(y \div B).Q$, and $P_3 \overset{\mathrm{def}}{=} z[y:\emptyset].Q$ do not exhibit a barb z, written $P_i \not\downarrow_z$ for $i = 1,2,3$. In contrast, when $x \neq z$ and $A \cap \{x\} \neq \emptyset$ we have that $(\mathsf{new}\, x)z[y:A].P\downarrow_z$, and when $x \neq z$ we have $[\mathsf{hide}\, x]z(y \div B).P$. Whenever $P \overset{\mathrm{def}}{=} [\mathsf{hide}\, y]\overline{x}\langle v\rangle.Q$ with $y \neq x$, we have $P\downarrow_{\overline{x}}$ if $y \neq v$, and $P\not\downarrow_{\overline{x}}$ otherwise. Weak barbs are defined by ignoring reductions. We let $P\Downarrow_x$ whenever $P \Rightarrow P'$ and $P'\downarrow_x$; similarly $P\Downarrow_{\overline{x}}$ whenever $P \Rightarrow P'$ and $P'\downarrow_{\overline{x}}$.

Following the standard definition of observational equivalence, we are aiming at an equivalence relation that is sensitive to the barbs, is closed under reduction, and is preserved by certain contexts.

$$\frac{z \notin B}{x(y \div B).P \xrightarrow{x(z)} P\{z/y\}} \qquad \frac{z \in A}{x[y : A].P \xrightarrow{x(z)} P\{z/y\}} \qquad \text{[L-}\textsc{In}\text{],[L-}\textsc{In-T}\text{]}$$

$$\frac{}{\overline{x}\langle y\rangle.P \xrightarrow{\overline{x}\langle y\rangle} P} \qquad \frac{P \xrightarrow{\overline{x}\langle y\rangle} P' \qquad y \neq x}{(\mathsf{new}\,y)P \xrightarrow{(y)\overline{x}\langle y\rangle} P'} \qquad \text{[L-}\textsc{Out}\text{],[L-}\textsc{Open}\text{]}$$

$$\frac{P \xrightarrow{x(y)} P' \qquad Q \xrightarrow{\overline{x}\langle y\rangle} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \qquad \text{[L-}\textsc{Com}\text{]}$$

$$\frac{P \xrightarrow{x(y)} P' \qquad Q \xrightarrow{(y)\overline{x}\langle y\rangle} Q' \qquad y \notin \mathrm{fn}(P)}{P \mid Q \xrightarrow{\tau} (\mathsf{new}\,y)(P' \mid Q')} \qquad \text{[L-}\textsc{Close}\text{]}$$

$$\frac{P \xrightarrow{\alpha} P' \qquad x \notin \mathrm{fn}(\alpha)}{(\mathsf{new}\,x)P \xrightarrow{\alpha} (\mathsf{new}\,x)P'} \qquad \frac{P \xrightarrow{\alpha} P' \qquad x \notin \mathrm{fn}(\alpha)}{[\mathsf{hide}\,x]P \xrightarrow{\alpha} [\mathsf{hide}\,x]P'} \qquad \text{[L-}\textsc{New}\text{],[L-}\textsc{Hide}\text{]}$$

$$\frac{P \xrightarrow{\alpha} P' \qquad \mathrm{bn}(\alpha) \cap \mathrm{fn}(Q) = \emptyset}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \qquad \frac{P \xrightarrow{\alpha} P'}{P \xrightarrow{\alpha} P' \mid !P} \qquad \text{[L-}\textsc{Par}\text{],[L-}\textsc{Repl}\text{]}$$

Figure 3: Labelled transition system

Definition 3.2 (Barb preservation)**.** *A relation $\mathscr{R}$ over processes is barb preserving if $P \mathscr{R} Q$, $P\downarrow_x$ implies $Q\Downarrow_x$, and $P\downarrow_{\overline{x}}$ implies $Q\Downarrow_{\overline{x}}$.*

The requirement of reduction closure is to ensure that the processes maintain their correspondence through the computation.

Definition 3.3 (Reduction closure)**.** *A relation $\mathscr{R}$ over processes is reduction-closed if $P \mathscr{R} Q$ and $P \rightarrow P'$ implies that $Q \Rightarrow Q'$ and $P' \mathscr{R} Q'$.*

We require contextuality with respect to the parallel composition, the new and the hide operators (cf. Section 2).

Definition 3.4 (Contextuality)**.** *A relation $\mathscr{R}$ over processes is contextual if $P \mathscr{R} Q$ implies $C[P] \mathscr{R} C[Q]$.*

Definition 3.5 (Observational equivalence)**.** *Observational equivalence, noted $\cong$, is the largest symmetric relation over processes which is barb preserving, reduction closed and contextual.*

Observational equivalence is difficult to establish since it requires quantification over contexts. In the next section we will introduce labelled transition semantics for the secret π-calculus, and show that the induced bisimulation coincides with observational equivalence. Besides the theoretical interest, this will be also of help in proving that two processes are observationally equivalent.

3.1 Characterization

The characterization relies on labelled transitions of the form $P \xrightarrow{\alpha} P'$, where α is one of the following actions:

$$\alpha = x(z) \mid \overline{x}\langle z\rangle \mid (z)\overline{x}\langle z\rangle \mid \tau$$

We let $\mathrm{fn}(x(z)) = \{x\}$, $\mathrm{fn}(\overline{x}\langle z\rangle) = \{x,z\}$, and $\mathrm{fn}\,(z)\overline{x}\langle z\rangle = \{x\}$. We define $\mathrm{bn}(x(z)) = \{z\}$, $\mathrm{bn}(\overline{x}\langle z\rangle) = \emptyset$ and $\mathrm{bn}((z)\overline{x}\langle z\rangle) = \{z\}$. We let $\mathrm{fn}(\tau) = \emptyset = \mathrm{bn}(\tau)$.

The transitions are defined by the rules in Figure 3. Action $x(z)$ represents the receiving of a name z on a channel x. In rule [L-IN], a process of the form $x(y \div B).P$ can receive a value z over x, provided that z is is not blocked ($z \notin B$). The received name will replace the formal parameter in the body of the continuation. Rule [L-IN-T] describes a trusted input, that is a process of the form $x[y : A].P$ that receives a variable z over x whenever z is accepted ($z \in A$); the variable z will replace all occurrences of y in P. The action $\overline{x}\langle y\rangle$ represents the output of a name y over x. This move is performed in [L-OUT] by the process $\overline{x}\langle y\rangle.P$ and leads to the continuation $P \triangleright B$. Communication arises in rule [L-COM] by means of a τ action obtained by a synchronization of an $x(y)$ action with a $\overline{x}\langle y\rangle$ action. Action $(y)\overline{x}\langle y\rangle$ is fired when the name y sent over x is bound by the new operator and its scope is opened by using rule [L-OPEN]. The scope of the new is closed by using rule [L-CLOSE]. In this rule the scope of a name y sent over x is enlarged to include a process which executes a dual action $x(y)$, giving rise to a synchronization of the two threads depicted by an action τ. Rule [L-NEW] is standard for restriction. Rule [L-HIDE] says that process $[\mathsf{hide}\,x]P$ performs an action α inferred from P, provided that the α does not contain x. Therefore extrusion of hidden channels is not possible, as previously discussed; note indeed that this the unique rule applicable for *hide*. Rule [L-REPL] performs a replication.

We have a standard notion of bisimilarity; in the following, we let $\overset{\tau}{\Longrightarrow}$ be the reflexive and transitive closure of $\xrightarrow{\tau}$.

Definition 3.6 (Bisimilarity)**.** *A symmetric relation $\mathscr{R}$ over processes is a bisimulation if whenever $P\mathscr{R}Q$ and $P \xrightarrow{\alpha} P'$ then there exists a process Q' such that $Q \overset{\tau}{\Longrightarrow} \xrightarrow{\hat{\alpha}} \overset{\tau}{\Longrightarrow} Q'$ and $P'\mathscr{R}Q'$ where $\hat{\tau}$ is the empty string and $\hat{\alpha} = \alpha$ otherwise. Bisimilarity, noted $\approx$, is the largest bisimulation.*

The following result establishes that bisimilarity can be used as a proof technique for observational equivalence; the proof is by coinduction and relies on the closure of bisimilarity under the new, hide and parallel composition operators.

Proposition 3.7 (Soundness)**.** *If $P \approx Q$ then $P \cong Q$.*

To prove the reverse direction, namely that behaviourally equivalent processes are bisimilar, we follow the approach of Hennessy [17] and proceed by co-induction relying on contexts C_α which emit the desired barbs whenever they interact with a process P such that $P \overset{\alpha}{\Longrightarrow} P'$, and vice versa. Perhaps interestingly, we can program a context to check if a given name is fresh even if our syntax does not include a matching construct (cf. [17, 8]). In Section 5 we will show that in the secret π-calculus the process $\mathsf{if}\, x = y\, \mathsf{then}\, P\, \mathsf{else}\, Q$ can be derived.

Proposition 3.8 (Completeness)**.** *If $P \cong Q$ then $P \approx Q$.*

Proof. Let $P\mathscr{R}Q$ whenever $P \cong Q$ and assume that $P \xrightarrow{\alpha} P'$. We show that there is Q' such that $Q \overset{\hat{\alpha}}{\Longrightarrow} Q'$ and $P' \equiv \mathscr{R} \equiv Q'$; this suffices to prove that $\mathscr{R}$ is included in observational equivalence (cf. [22]). Whenever $\alpha = \tau$, we use reduction-closure of $\cong$ to find Q' such that $Q \Longrightarrow Q'$ with $P' \cong Q'$. By relyng on a lemma that establishes that reductions correspond to τ actions, we infer that $Q \overset{\tau}{\Longrightarrow} Q'$, which is the desired result since $P'\mathscr{R}Q'$. Otherwise assume $\alpha \neq \tau$. We exploit contextuality of $\cong$ and infer that $C^A_\alpha[P] \cong C^A_\alpha[Q]$ where we let $A = \mathrm{fn}(P) \cup \mathrm{fn}(Q)$ and C^A_α be defined below. We let $A = \{a_1, \ldots, a_n\}$,

$I = 1, \dots, n$, with $n \geq 1$, and assume names $\omega, \psi_1, \dots, \psi_n$ such that $\{\omega, \psi_1, \dots, \psi_n\} \cap A = \emptyset$.

$$
\begin{array}{rcll}
C^A_{x(y)}[-] & \stackrel{\text{def}}{=} & - \mid \overline{x}\langle y\rangle.\overline{\omega}\langle\rangle & \\
C^A_{\alpha}[-] & \stackrel{\text{def}}{=} & - \mid [\mathsf{hide}\, k][x(z).(z[w:k] \mid \overline{\omega}\langle\rangle) \mid_{i\in I} \overline{a_i}\langle k\rangle.\overline{\psi_i}\langle\rangle] & \alpha = x\langle y\rangle, (y)x\langle y\rangle \\
C'_y & \stackrel{\text{def}}{=} & [\mathsf{hide}\, k][y[w:k] \mid \overline{\omega}\langle\rangle \mid_{i\in I} \overline{a_i}\langle k\rangle.\overline{\psi_i}\langle\rangle] & \forall i \in I \,.\, y \neq a_i \\
C''_y & \stackrel{\text{def}}{=} & [\mathsf{hide}\, k][\overline{\omega}\langle\rangle \mid \overline{\psi_l}\langle\rangle \mid_{i\in I\setminus l} \overline{a_i}\langle k\rangle.\overline{\psi_i}\langle\rangle] & a_l = y
\end{array}
$$

Assume $\alpha = \overline{x}\langle y\rangle$. We have that there is $a_l \in A, a_l = y$ such that $C^A_\alpha[P] \Longrightarrow\ \equiv C_P \stackrel{\text{def}}{=} P' \mid C''_y$. We find a process C_Q such that $C^A_\alpha[Q] \Longrightarrow C_Q \cong C_P$. Since $C_P \downarrow_{\bar{\omega}}, \downarrow_{\bar{\psi}_l}$, this implies that $C_Q \Downarrow_{\bar{\omega}}, \Downarrow_{\bar{\psi}_l}$. Therefore the weak barb $\bar{\omega}$ of C_Q has been unblocked since Q emits a weak action α' with x as subject. Moreover, the object of α' is y, that is $\alpha' = \alpha$, because of the weak barb $\bar{\psi}_l$. Indeed the thread $\overline{a_l}\langle k\rangle.\overline{\psi_l}\langle\rangle$ with $a_l = y$ can be unblocked only by $y[w:k]$, because k is protected by the hide declaration. Therefore there is Q' such that $Q \stackrel{\alpha}{\Longrightarrow} Q'$ and $C_Q \cong Q' \mid C''_y$. We conclude by showing that this implies $P' \cong Q'$, and in turn $P' \mathscr{R} Q'$, as requested. Assume $\alpha = (y)x\langle y\rangle$. We have that $C^A_\alpha[P] \Longrightarrow\ \equiv C_P \stackrel{\text{def}}{=} P' \mid C'_y$. Since y is fresh we have that $a_i \neq y$ for all $a_i \in A$. Therefore $C_P \not\Downarrow_{\bar{\psi}_i}$ for all $i \in I$, because k is protected by hide. We easily obtain that there is C_Q such that $C^A_\alpha[Q] \Longrightarrow C_Q \cong C_P$ with $C_Q \Downarrow_{\bar{\omega}}, \not\Downarrow_{\bar{\psi}_i}$, for all $i \in I$. This let us infer that there is Q' such that $Q \stackrel{\alpha}{\Longrightarrow} Q'$ and $C_Q \cong Q' \mid C'_y$, and the result then follows by showing that $P' \mid C'_y \cong Q' \mid C'_y$ implies $P' \cong Q'$. □

Full abstraction is obtained by Propositions 3.7 and 3.8.

Theorem 3.9 (Full Abstraction)**.** $\cong = \approx$.

4 Distrusting communications protected by restriction

In this section we introduce a *spy* process that represents a side-channel attack against communications that occur on untrusted channels, that is: channels that are not protected by hide. We assume that the spy is not able to retrieve the content of an exchange. The spy abstraction models the ability of the context to detect interactions when the processes are implemented by means of network protocols which do not rely on dedicated channels, and therefore require some mechanism to enforce the secrecy of the message (e.g. cryptography). This ability leads to break some of the standard security equations for the new operator, which can be recovered by re-programming the protocol and making use of the hide operator. We add to the syntax of the secret π-calculus the following process where we let spy be a reserved keyword. We let P, Q, R to range over *spied processes*.

$$
\begin{array}{rcll}
P, Q, R & ::= & \cdots \mid \mathsf{spy} : S.P & \quad\text{spied processes} \\
S & ::= & \{x\} \mid \emptyset & \quad\text{spied set}
\end{array}
$$

When in $\mathsf{spy} : S.P$ the spied set S is equal to $\{x\}$, noted $\mathsf{spy} : x.P$, this permits to make explicit which (free) reduction the spy shall observe. Note that listening on multiple names can be easily programmed by putting in parallel several spies. The spy process $\mathsf{spy} : \emptyset.P$, noted $\mathsf{spy}.P$, will be used to detect reductions protected by restriction. We let the free and bound names of the *spy* be defined as follows: $\mathrm{fn}(\mathsf{spy} : S.R) \stackrel{\text{def}}{=} S \cup \mathrm{fn}(R)$ and $\mathrm{bn}(\mathsf{spy} : S.R) \stackrel{\text{def}}{=} \mathrm{bn}(R)$.

New rules for blocking a name

$$(\mathsf{spy} : S.P) \uplus b \equiv \mathsf{spy} : S.(P \uplus b)$$

New rules for structural congruence

$$\begin{array}{ll}
(\mathsf{new}\,x)(P) \mid \mathsf{spy}.R \equiv (\mathsf{new}\,x)(P \mid \mathsf{spy} : x.R) & x \notin \mathrm{fn}(\mathsf{spy}.R) \\
(\mathsf{new}\,x)(P) \mid \mathsf{spy} : y.R \equiv (\mathsf{new}\,x)(P \mid \mathsf{spy} : y.R) & x \notin \mathrm{fn}(\mathsf{spy} : y.R) \\
[\mathsf{hide}\,x][P] \mid \mathsf{spy} : S.R \equiv [\mathsf{hide}\,x][P \mid (\mathsf{spy} : S.R) \uplus x] & x \notin \mathrm{fn}(\mathsf{spy} : S.R)
\end{array}$$

New reduction rules

$$\frac{z \notin B}{x(y \div B).P \mid \overline{x}\langle z\rangle.Q \mid \mathsf{spy} : x.R \rightarrow P\{z/y\} \mid Q \mid R} \qquad [\text{RS-COM}]$$

$$\frac{z \in A}{x[y : A].P \mid \overline{x}\langle z\rangle.Q \mid \mathsf{spy} : x.R \rightarrow P\{z/y\} \mid Q \mid R} \qquad [\text{RS-T-COM}]$$

Figure 4: Spied process semantics

The semantics of spied processes is described by adding the communication rules in Figure 4 to those in Figure 2: The rules describe a form of synchronization among three processes: a sender on channel x, a receiver on channel x, and a *spy* on channel x. More in detail, rule [RS-COM] depicts a synchronization among an input of the form $x(y \div B).P$, a sender and a spy, while rule [RS-T-COM] describes a similar three-synchronization but for a trusted input of the form $x[y : A].P$.

The definition of observational equivalence for spied processes is obtained by extending Definition 3.5 to the semantics in Figure 4; we indicate the resulting equivalence with $\overset{\bullet}{\cong}$. This will permit to study the security of processes in presence of the *spy*.

To make the picture clear, in Figure 5 we introduce labelled transition semantics for spied processes. We consider two new actions $?x$ and $!x$ corresponding respectively to the presence of a *spy* and to a signal of communication.

$$\alpha \ ::= \ \cdots \mid ?x \mid !x$$

We assume the existence of variable $\nu \in \mathscr{N}$ that cannot occur in the process syntax, and we use it to signal restricted communications. It is convenient to define the notion of (free) subject and object of an action. We let $\mathrm{subj}(\alpha) \overset{\text{def}}{=} \{x\}$ whenever $\alpha = \overline{x}\langle y\rangle, (y)\overline{x}\langle y\rangle, x(y)$, and be empty otherwise. We define $\mathrm{obj}(\alpha) \overset{\text{def}}{=} \{y\}$ whenever $\alpha = \overline{x}\langle y\rangle, x(y)$, and $\mathrm{obj}(\alpha) = \emptyset$ otherwise.

The lts in Figure 5 introduces three new rules for the *spy*, [L-SPY], [L-SPY-RES] and [L-SPY-COM], and re-defines the rules for restriction, for hide and for communication of Figure 3. In rule [L-SPY] the process $\mathsf{spy} : x.P$ can fire an action $?x$ and progress to P. The dual action, $!x$, is fired in rules [L-COM] and [L-CLOSE] whenever a communication occurred on a free channel x. Rule [L-SPY-COM] describes the eaves-dropping of a communication. A process of the form $\mathsf{spy}.P$ can only fire an action $?\nu$ through rule [L-SPY-RES]. In rule [L-NEW] we use a partial function $(\!|\cdot|\!)_x$ to relabel the action fired underneath a restriction: we let $(\!|\alpha|\!)_x \overset{\text{def}}{=} \alpha$ whenever $x \notin \mathrm{fn}(\alpha)$, $(\!|!x|\!)_x \overset{\text{def}}{=} !\nu$, $(\!|?x|\!)_x \overset{\text{def}}{=} ?\nu$. This will be used to signal restricted communications, as introduced. Differently, in rule [L-HIDE] we use a relabeling partial

$$\frac{}{\mathsf{spy} : x.P \xrightarrow{?x} P} \qquad \frac{}{\mathsf{spy}.P \xrightarrow{?v} P} \qquad \text{[L-Spy],[L-Spy-Res]}$$

$$\frac{P \xrightarrow{x(y)} P' \qquad Q \xrightarrow{\overline{x}\langle y\rangle} Q'}{P \mid Q \xrightarrow{!x} P' \mid Q'} \qquad \frac{P \xrightarrow{x(y)} P' \qquad Q \xrightarrow{(y)\overline{x}\langle y\rangle} Q' \qquad y \notin \mathrm{fn}(P)}{P \mid Q \xrightarrow{!x} (\mathsf{new}\,y)(P' \mid Q')} \qquad \text{[L-Com],[L-Close]}$$

$$\frac{P \xrightarrow{!x} P' \qquad Q \xrightarrow{?x} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \qquad \text{[L-Spy-Com]}$$

$$\frac{P \xrightarrow{\alpha} P' \qquad x \notin \mathsf{subj}(\alpha)}{(\mathsf{new}\,x)P \xrightarrow{(\!|\alpha|\!)_x} (\mathsf{new}\,x)P'} \qquad \frac{P \xrightarrow{!x} P' \qquad x \notin \mathsf{subj}(\alpha) \cup \mathsf{obj}(\alpha)}{[\mathsf{hide}\,x]P \xrightarrow{[\![\alpha]\!]_x} [\mathsf{hide}\,x]P'} \qquad \text{[L-New],[L-Hide]}$$

Figure 5: Labelled transitions for spied processes

function $[\![\cdot]\!]_x$ that makes invisible communications that occur under *hide*. We let $[\![\alpha]\!]_x \overset{\text{def}}{=} \alpha$ whenever $x \notin \mathrm{fn}(\alpha)$, $[\![!x]\!]_x \overset{\text{def}}{=} \tau$ and $[\![?x]\!]_x \overset{\text{def}}{=} \tau$.

Definition 4.1 (Bisimilarity). *A symmetric relation $\mathscr{R}$ over spied processes is a bisimulation if whenever $R_1 \mathscr{R} R_2$ and $R_1 \xrightarrow{\alpha} R'$ then there exists a spied process R'' such that $R_2 \overset{\tau}{\Longrightarrow} \xrightarrow{\hat{\alpha}} \overset{\tau}{\Longrightarrow} R''$ and $R' \mathscr{R} R''$ where $\hat{\tau}$ is the empty string, and $\hat{\alpha} = \alpha$ otherwise. Bisimilarity, noted $\overset{\bullet}{\approx}$, is the largest bisimulation.*

By using the same construction of Section 3.1, we obtain the main result of this section: observational equivalence for spied processes and bisimilarity coincide. As a by-product, we can also use bisimulation as a technique to prove that two processes cannot be distinguished by the *spy*.

Theorem 4.2 (Full Abstraction). $\overset{\bullet}{\cong} = \overset{\bullet}{\approx}$.

Sketch of the proof. To see that behavioural equivalence is included in bisimilarity, we proceed by co-induction as in the proof of Proposition 3.8 by relying on contexts C^A_α that detect whenever a process does emit a weak action α. Given a set of names A such that $\mathrm{fn}(\alpha) \subseteq A$ and $\omega \notin A$ we define the following contexts to account for the new actions $!x$ and $?x$.

$$\begin{aligned}
C^A_{!x}[-] &\overset{\text{def}}{=} \mathsf{spy} : x.\overline{\omega}\langle\rangle && x \neq v\\
C^A_{?x}[-] &\overset{\text{def}}{=} x(y).\overline{\omega}\langle\rangle \mid \overline{x}\langle\rangle && x \neq v\\
C^A_{!v}[-] &\overset{\text{def}}{=} \mathsf{spy}.\overline{\omega}\langle\rangle\\
C^A_{?v}[-] &\overset{\text{def}}{=} (\mathsf{new}\,x)(x(y).\overline{\omega}\langle\rangle \mid \overline{x}\langle\rangle)
\end{aligned}$$

The proof then proceeds routinely by following a schema similar to the one of Proposition 3.8. The reverse direction, namely that bisimilarity is contained in behavioural equivalence, is shown by proving that $\overset{\bullet}{\approx}$ is closed under the new, hide, and parallel composition operators. See [15] for all the details. □

5 Properties of the secret π-calculus

In this section we discuss some algebraic properties of the secret π-calculus, and we show how we can implement the name matching operator. Lastly we provide an example of deployment of a mandatory access control policy that is inspired by the D-Bus technology [21]. In the following, we write $P \overset{\bullet}{\ncong} Q$ to indicate that $(P,Q) \notin \overset{\bullet}{\cong}$. We also write $\overline{x}\langle\rangle$ and omit to indicate the message in output whenever this is irrelevant, and use the notation $[\mathsf{hide}\, B]P$ to indicate the process $[\mathsf{hide}\, b_1] \cdots [\mathsf{hide}\, b_n]P$ whenever $B = \{b_1, \dots, b_n\}$.

Algebraic equalities and inequalities The first inequality illustrates the mechanism of blocked names.

$$x(y \div B).P \overset{\bullet}{\ncong} x(y \div B').P \qquad\qquad B \neq B' \tag{2}$$

To prove (3) let $z \in B'$, $z \notin B$ and consider the context $C[-] \overset{\text{def}}{=} [\mathsf{hide}\, B, B'][\overline{x}\langle z\rangle.\overline{\varpi}\langle\rangle \mid -]$ with ω free, $\omega \notin \mathrm{fn}(P)$. By applying [R-COM] followed by applications of [R-HIDE] we have that $C[x(y \div B).P] \to [\mathsf{hide}\, B, B'][\overline{\varpi}\langle\rangle \mid P\{z/y\}]$, that is $C[x(y).P] \Downarrow_{\bar{\omega}}$. In contrast, we have that $C[x(y \div B').P] \not\Downarrow_{\bar{\omega}}$, because of $z \in B'$. The case $B' \subseteq B$ is analogous.

We have a similar result for accepted names.

$$x[y : A].P \overset{\bullet}{\ncong} x[y : A'].P \qquad\qquad A \neq A' \tag{3}$$

A distinguishing context is $C[-] \overset{\text{def}}{=} \overline{x}\langle a\rangle.\overline{\varpi}\langle\rangle \mid -$ where ω is fresh and $a \in A, a \notin A'$ if $A \not\subset A'$, and $a \in A', a \notin A$ otherwise.

The next inequality illustrates the discriminating power of the *spy*.

$$(\mathsf{new}\, x)(\overline{x}\langle z\rangle \mid x(y)) \overset{\bullet}{\ncong} \mathbf{0} \tag{4}$$

To prove (4), consider the context $C[-] = \mathsf{spy}.\overline{\varpi}\langle\rangle \mid -$. By applying [RS-COM] and [R-NEW] followed by [R-STRUCT] we infer $C[(\mathsf{new}\, x)(\overline{x}\langle y\rangle \mid x(y))] \to \overline{\varpi}\langle\rangle$: that is, $C[(\mathsf{new}\, x)(\overline{x}\langle y\rangle \mid x(y))] \Downarrow_{\bar{\omega}}$ while $C[\mathbf{0}] \not\Downarrow_{\bar{\omega}}$.

The invisibility of communications protected by using the *hide* operator is established by means of the equation below, which is proved by co-induction.

$$[\mathsf{hide}\, x][\overline{x}\langle z\rangle \mid x(y).Q] \overset{\bullet}{\cong} [\mathsf{hide}\, x][Q\{z/y\}] \tag{5}$$

The last equation states the impossibility of extrusion of hidden channels.

$$[\mathsf{hide}\, x][\overline{z}\langle x\rangle] \overset{\bullet}{\cong} \mathbf{0} \tag{6}$$

Implementing name matching Name matching is not needed as an operator in our calculus (cf. [12]). We show this by providing a semantics-preserving translation of the if-then-else construct [17]. Consider the process $\mathsf{if}\, x = y\, \mathsf{then}\, P\, \mathsf{else}\, Q$ which reduces to P whenever $x = y$, and reduces to Q otherwise. Let $Z \overset{\text{def}}{=} \mathrm{fn}(\mathsf{if}\, x = y\, \mathsf{then}\, P\, \mathsf{else}\, Q)$; therefore there are names $z_1, \dots, z_n$, $n \geq 0$, s.t. $Z = \{x, z_1, \dots, z_n\}$. Let $I = \{1, \dots, n\}$ and assume k fresh. We define:

$$[\![\mathsf{if}\, x = y\, \mathsf{then}\, P\, \mathsf{else}\, Q]\!]_Z \overset{\text{def}}{=} [\mathsf{hide}\, k][y[w : k] \mid \overline{x}\langle k\rangle.(P \uplus k) \mid_I \overline{z_i}\langle k\rangle.(Q \uplus k)]$$

Whenever $x = y$, we have that the only possible reduction arises among the trusted input $y[w : k]$ and $\overline{x}\langle k\rangle.(P \uplus k)$, leading to $P' \stackrel{\text{def}}{=} [\mathsf{hide}\, k][P \uplus k \,|_I\, \overline{z_i}\langle k\rangle.(Q \uplus k)]$. Note that P and P' have the same interactions with the context, because k is blocked in all threads of P': therefore Q cannot be unblocked. This result can be formalized by relying on the behavioural theory [1] of the secret π-calculus.
We infer the following equation:

$$[\![\mathsf{if}\, x = x\, \mathsf{then}\, P\, \mathsf{else}\, Q]\!]_Z \cong P \tag{7}$$

Consider now the case $x \neq y$ and let $y = z_1$. The matching process reduces to the rearranged process $[\mathsf{hide}\, k][\overline{x}\langle k\rangle.(P \uplus k) \mid Q \uplus k \,|_{\{2,\dots,n\}}\, \overline{z_i}\langle k\rangle(Q \uplus k)]$, which has the same behaviour of Q:

$$[\![\mathsf{if}\, x = y\, \mathsf{then}\, P\, \mathsf{else}\, Q]\!]_Z \cong Q \qquad x \neq y \tag{8}$$

Modeling dedicated channels Security mechanisms based on dedicated channels can be naturally modeled in the secret π-calculus. D-Bus [21] is an IPC system for software applications that is used in many desktop environments. Applications of each user share a private bus for asynchronous message-passing communication; a system bus permits to broadcast messages among applications of different users. Versions smaller than 0.36 contain an erroneous access policy for channels which allows users to send and listen to messages on another user's channel if the address of the socket is known. We model this vulnerability by means of an *internal* attacker that leaks the user's channel. In the specification below, two applications of an user U_1 utilize a private bus to exchange a password; in fact, the password can be intercepted by the user U_2 through the malicious code $!\overline{sys}\langle c\rangle$ of U_1, which publishes c on the system bus.

$$U_1 \stackrel{\text{def}}{=} (\mathsf{new}\, c)(!\overline{sys}\langle c\rangle \mid (\mathsf{new}\, pwd)\overline{c}\langle pwd\rangle \mid c(x).P) \qquad\qquad U_2 \stackrel{\text{def}}{=} sys(x).x(y_{pwd}).Q \tag{9}$$

The patch released by Fedora restricts the access to the user's bus: only applications with the same user-id can have access. We stress that this policy is mandatory: that is, the user cannot change it. By using the secret π-calculus we can easily patch U_1 by hiding the bus: $U' \stackrel{\text{def}}{=} [\mathsf{hide}\, c][!\overline{sys}\langle c\rangle \mid (\mathsf{new}\, pwd)(\overline{c}\langle pwd\rangle) \mid c(x).P]$. The following equation, which can be proved co-inductively, states that the policy is fulfilled even in presence of internal attacks:

$$U' \overset{\bullet}{\cong} [\mathsf{hide}\, c][(\mathsf{new}\, pwd)(P\{pwd/x\})] \tag{10}$$

6 Related work

Many analysis and programming techniques for security have been developed for process calculi. Among these, we would mention the security analysis enforced by means of static and dynamic type-checking (e.g. [13, 16, 10]), the verification of secure implementations and protocols that are protected by cryptographic encryption (e.g. [7, 4, 2, 11]), and programming models that consider a notion of location (e.g. [17, 24, 14]).

The paper [13] introduces a type system for a π-calculus with groups that permits to control the distribution of resources: names can be received only by processes in the scope of the group. The intent

[1] Note that observational equivalence is not preserved by input-prefixing; the outlined translation could be indeed sensitive to name aliasing.

is, as in our paper, to preserve the accidental or malicious leakage of secrets, even in the presence of untyped opponents. A limitation of [13] is that processes that are not statically type-checked are interpreted as opponents trying to leak secrets. On contrast, our aim is to consider systems where processes could dynamically join the system at run-time; this permits us to analyze the secrecy of protocols composed by trusted sub-systems that can grow in size of the number of the participants. While devising an algorithm for type checking groups can be non-trivial (cf. [25]), we note that actual systems do not often rely on types, even for local communications. For instance D-Bus (cf. Section 5) relies on a mandatory access control policy enforced at the kernel level through process IDs. Our semantics-based approach appears as adequate to describe such low-level mechanisms.

As discussed in the introduction, concrete implementations of π-calculi models do protect communications by means of cryptography. The problem of devising a secure, fully abstract implementation has been first introduced in [1] and subsequently tackled for the join calculus in [4]. The paper [7] introduces a bisimulation-based technique to prove equivalences of processes using cryptographic primitives; this can be used to show that a protocol does preserve secrecy. We follow a similar approach and devise bisimulation semantics for establishing the secrecy of processes running in an environment where the distribution of channels is controlled. The presence of a spy in our model is reminiscent of the network abstraction of [9]. In that paper, the network provides the low-level counter part of the model where attacks based on bit-string representations, interception, and forward/reply can be formalized.

From the language design point of view, we share some similarity with the ideas behind the boxed π-calculus [24]. A box in [24] acts as wrapper where we can confine untrusted process; communication among the box and the context is subject to a fine-grained control that prevents the untrusted process to harm the protocol. Our hide operator is based on the symmetric principle: processes within the scope of an hide can run their protocol without be disturbed by the context outside it.

An interesting approach related to ours in spirit – but not in conception or details – is D-fusion [6]. The calculus has two forms of restriction: A ”ν” operator for name generation, and a ”λ” operator that behaves like an existential quantifier and it can be seen as a generalization of an input binder. Both operators allow extrusion of the entities they declare but only the former guarantees uniqueness. In contrast our hide operator is not meant as an existential nor as an input-binder and it prevents the extrusion of the name it declares.

Acknowledgements We wholeheartedly thank the extremely competent, anonymous reviewers of EXPRESS 2012. They went beyond the call of duty in providing excellent reports which have been very helpful to improve our paper.

References

[1] Martín Abadi (1998): *Protection in Programming-Language Translations*. In: *ICALP, LNCS* 1443, Springer, pp. 868–883, doi:10.1007/BFb0055109.

[2] Martín Abadi, Bruno Blanchet & Cédric Fournet (2007): *Just fast keying in the pi calculus*. *ACM Trans. Inf. Syst. Secur.* 10(3), doi:10.1145/1266977.1266978.

[3] Martín Abadi & Cédric Fournet (2001): *Mobile values, new names, and secure communication*. In: *POPL*, ACM press, pp. 104–115, doi:10.1145/360204.360213.

[4] Martín Abadi, Cédric Fournet & Georges Gonthier (2002): *Secure Implementation of Channel Abstractions*. *Inf. Comput.* 174(1), pp. 37–83, doi:10.1006/inco.2002.3086.

[5] Martín Abadi & Andrew D. Gordon (1999): *A Calculus for Cryptographic Protocols: The spi Calculus*. *Inf. Comput.* 148(1), pp. 1–70, doi:10.1006/inco.1998.2740.

[6] Michele Boreale, Maria Grazia Buscemi & Ugo Montanari (2004): *D-Fusion: A Distinctive Fusion Calculus*. In: *APLAS*, pp. 296–310, doi:10.1007/978-3-540-30477-7_20.

[7] Michele Boreale, Rocco De Nicola & Rosario Pugliese (2001): *Proof Techniques for Cryptographic Processes*. *SIAM J. Comput.* 31(3), pp. 947–986, doi:10.1137/S0097539700377864.

[8] Michele Boreale & Davide Sangiorgi (1998): *Bisimulation in Name-Passing Calculi without Matching*. In: *LICS*, IEEE Computer Society, pp. 165–175, doi:10.1109/LICS.1998.705653.

[9] Michele Bugliesi & Riccardo Focardi (2010): *Channel abstractions for network security*. *Mathematical Structures in Computer Science* 20(1), pp. 3–44, doi:10.1017/S0960129509990247.

[10] Michele Bugliesi & Marco Giunti (2005): *Typed Processes in Untyped Contexts*. In: *TGC*, *LNCS* 3705, Springer, pp. 19–32, doi:10.1007/11580850_3.

[11] Michele Bugliesi & Marco Giunti (2007): *Secure implementations of typed channel abstractions*. In: *POPL*, ACM press, pp. 251–262, doi:10.1145/1190216.1190253.

[12] Marco Carbone & Sergio Maffeis (2003): *On the Expressive Power of Polyadic Synchronisation in pi-calculus*. *Nord. J. Comput.* 10(2), pp. 70–98, doi:10.1016/S1571-0661(05)80361-5.

[13] Luca Cardelli, Giorgio Ghelli & Andrew D. Gordon (2005): *Secrecy and group creation*. *Inf. Comput.* 196(2), pp. 127–155, doi:10.1016/j.ic.2004.08.003.

[14] Giuseppe Castagna, Jan Vitek & Francesco Zappa Nardelli (2005): *The Seal Calculus*. *Inf. Comput.* 201(1), pp. 1–54, doi:10.1016/j.ic.2004.11.005.

[15] Marco Giunti, Catuscia Palamidessi & Frank D. Valencia: *Hide and New in the π-calculus*. Available at `http://www.lix.polytechnique.fr/~marco.giunti`. Long version of this paper.

[16] Matthew Hennessy (2005): *The security pi-calculus and non-interference*. *J. Log. Algebr. Program.* 63(1), pp. 3–34, doi:10.1016/j.jlap.2004.01.003.

[17] Matthew Hennessy (2007): *A Distributed Pi-Calculus*. Cambridge University Press, New York, NY, USA.

[18] R. Milner (1980): *A Calculus of Communicating Systems*. *LNCS* 92, Springer-Verlag.

[19] R. Milner, J. Parrow & D. Walker (1992): *A Calculus of Mobile Processes, part I*. *Inf. Comput.* 100(1), pp. 1–40, doi:10.1016/0890-5401(92)90008-4.

[20] R. Milner, J. Parrow & D. Walker (1992): *A Calculus of Mobile Processes, part II*. *Inf. Comput.* 100(1), pp. 41–77, doi:10.1016/0890-5401(92)90009-5.

[21] Havoc Pennington, Anders Carlsson, Alexander Larsson, Sven Herzberg, Simon McVittie & David Zeuthen: *D-Bus Specification*. Available at `http://dbus.freedesktop.org`.

[22] Davide Sangiorgi & Robin Milner (1992): *The Problem of "Weak Bisimulation up to"*. In: *CONCUR*, *LNCS* 630, Springer, pp. 32–46, doi:10.1007/BFb0084781.

[23] Davide Sangiorgi & David Walker (2001): *The pi-calculus, a theory of mobile processes*. Cambridge University Press.

[24] Peter Sewell & Jan Vitek (2003): *Secure Composition of Untrusted Code: Box pi, Wrappers, and Causality*. *J. Comp. Sec.* 11(2), pp. 135–188. Available at `http://iospress.metapress.com/content/6u3ue7xblwqprxhx/`.

[25] Vasco T. Vasconcelos & Kohei Honda (1993): *Principal Typing Schemes in a Polyadic pi-Calculus*. In: *CONCUR*, *LNCS* 715, Springer, pp. 524–538, doi:10.1007/3-540-57208-2_36.