Secrecy in Untrusted Networks*

Michele Bugliesi[1], Silvia Crafa[1], Amela Prelic[2], and Vladimiro Sassone[3]

Abstract. We investigate the protection of migrating agents against the untrusted sites they traverse. The resulting calculus provides a formal framework to reason about protection policies and security protocols over distributed, mobile infrastructures, and aims to stand to ambients as the spi calculus stands to π. We present a type system that separates trusted and untrusted data and code, while allowing safe interactions with untrusted sites. We prove that the type system enforces a privacy property, and show the expressiveness of the calculus via examples and an encoding of the spi calculus.

Introduction

Secure communication in the π-calculus relies on *private channels*. Process

$$(\nu n)(\ \overline{n}\langle m\rangle \mid n(x).P\ ) \tag{1}$$

uses a private channel n to transmit message m. Intuitively, this guarantees the secrecy of m since no third process may interfere with n. In a distributed network, however, the subprocesses $\overline{n}\langle m\rangle$ and $n(x).P$ may be located at remote sites, and the link between these be physically insecure, regardless of the privacy of n. It may therefore be desirable to implement a channel meant to deliver private information with lower level mechanisms. For instance, as an *encrypted* connection over a *public* channel, as in the spi calculus [3]:

$$(\nu n)(\ \overline{p}\langle\{m\}_n\rangle \mid p(y).case\ y\ of\ \{x\}_n\ in\ P\ ) \tag{2}$$

The knowledge of n is still confined here, but its role is different: an encryption key, rather than a channel. The message is encrypted and communicated along a public channel p; even though the encrypted packet is intercepted, only the intended receivers, which possess the key n, may decrypt it to obtain m (cf. [1] for a thorough discussion on the weaknesses of the scheme.)

Similar mechanisms for secrecy are available for Mobile Ambients (MA). The following process, for instance, provides for the exchange of messages between locations a and b.

$$(\nu n)(a[\ n[\mathsf{out}\ a.\mathsf{in}\ b.\langle m\rangle]\ ] \mid b[\mathsf{open}\ n.(x).P])\tag{3}$$

Ideally, no adversary can discover or seize m, or cause a different message to be delivered at b, as m is encapsulated into the secret ambient n.

The question we address in this paper is whether the abstract enveloping mechanism above can be turned into a realistic model of security for calculi of mobile agents that need to enforce protection policies and secrecy guarantees in untrusted environments. The answer we provide is articulated, and leads us to introduce more flexible, and lower-level mechanisms. Our work parallels the development which brought from π to spi, in the ambition of identifying suitable such primitives.

* Research supported by 'MyThS: Models and Types for Security in Mobile Distributed Systems', EU FET-GC IST-2001-32617. (1) Università "Ca' Foscari", Venezia; (2) Max-Planck-Institut für Informatik; (3) University of Sussex

Structure of the paper. §1 discusses how to achieve secrecy in (variants of) MA, and presents the motivations for introducing specific primitives, while §2 provides a formal definition of the outcome, the SBA calculus, and illustrates it with a few examples. A key point of our work is the development in §3 of a type system which accounts for interactions between trusted and untrusted (*opponents*) components traveling over open networks. Types split the world in two: the trusted *system* and the untrusted *context*. Relying on such intuition, any data coming from the external environment is assigned an "unknown" type Public; and Public values are then handled with suspicion, since there is no saying what they are, or whom they are from. The type system guarantees a fundamental property of *secrecy* proved in §4: a well typed process does not disclose its secrets to any adversary, even though these may know its public names and may traverse its sub-ambients. §5 presents an encoding of the spi calculus in SBA, which we consider as the starting point for a future deeper comparison between spi and SBA. A further section concludes the presentation with final remarks.

1 A Sealing Mechanism for Ambients

The literature on mobile agent security focuses mainly on the dual problems of protecting a host from incoming agents, and protecting a mobile agent from malicious hosts. Cryptography is used effectively in the latter case, by setting up a network of trusted sites and a mechanism of authentication between such sites and encrypted agents on the move (cf. [20,21,18]). The sealing mechanisms we envision aim instead at protecting the secrecy of data inside ambients which move freely in a network of possibly unreliable sites. The first question is whether these mechanisms are needed at all.

The security model of the Ambient Calculus [8] is centered around the idea that names provide the key to access the contents (data and code) encapsulated by ambients. Accordingly, as long an ambient name is secret, the ambient content is protected from any undesired access. The protocol for message exchange in (3) above is based on this assumption.

Our approach here questions this assumption, based on the observation that ambient movement cannot occur without ambients revealing their names to some (not necessarily trusted) component of the underlying infrastructure. Besides happening in current implementations (cf. the ambient managers of [12] and the pointers-to-parent of [19]), it is hard to conceive how it could possibly be otherwise. In the internetworking of the near future, crossing boundaries (routers, gateways, firewalls, ...) will involve running complex protocols. Traveling active packets will have to negotiate several conditions, such as QoS guarantees and bandwidth occupation, as well as paying for the service received. The principles of interoperability across different networks and of data encapsulation will impose such protocols to work as direct dialogs between the interested parties, which can only rely on direct communication and, therefore, force agents to reveal their names to the network. Thus, quite similarly to the fact that secure remote communication cannot rely exclusively on private names, the security of a mobile ambient cannot be relegated to the confidentiality of its name. Then, back to our example, the encapsulation mechanism of (3) turns out not to be secure, as in a realistic scenario name n will have to be disclosed.

We may think of two alternative ways to provide for stronger security guarantees.

- ▷ One solution is to commit to agents their own security, by resorting to co-capabilities. For instance, process (2) can be recast in Safe Ambients [15] as shown below.

$$(\nu n)(a[\,n[\,\mathsf{out}\ a.\mathsf{in}\ b.\overline{\mathsf{open}}\ n.\langle m\rangle\,]\ \mid \overline{\mathsf{out}}\ \ a\,]\ \mid b[\,\overline{\mathsf{in}}\ b.\mathsf{open}\ n.(x)P\,]\,) \tag{4}$$

- ▷ An alternative solution is to protect the secrecy of an ambient name by a wrapping the ambient in a box that carries it to destination:

$$(\nu n)(a[\,p[\,\mathsf{out}\ a.\mathsf{in}\ b \mid n[\,\langle m\rangle\,]\,]\,] \mid b[\,\mathsf{open}\ p.\mathsf{open}\ n.(x)P\,]) \tag{5}$$

The first protocol guarantees that no one can enter n, or open it and read m before n reaches b, even though the name n is revealed while n is on the move. In the second protocol, name n and message m are protected by the wrapper ambient p, which can only be opened at the target site b. Now n need not reveal its name because it does not move.

Whether or not these solutions are indeed satisfactory depends on the kind of agents and networks at which our formal calculus is targeted.

If we take ambients as an abstraction of physical devices, such as laptops or PDA's, then the first approach is likely to be all we need: physical devices can easily perform access control to protect their contents in ways similar to those encompassed by the use of co-capabilities exemplified in (4). If instead, we think of ambients as representations of "soft" agents, then the first approach is only appropriate in rather "friendly" networks, whose gateways respect the privacy of the code they route to the next hop to destination.

The second approach, instead, is more robust and applies well to the case of soft agents. In particular, in (5), we may think of $n[\,\langle m\rangle\,]$ as a piece of data encrypted under the key n: this is consistent with the structure of the protocol, as ambient n need not be active while inside p, since it is the thread $\mathsf{out}\ a.\mathsf{in}\ b$ that routes p (hence n) to destination. On the other hand, this solution cannot be fruitfully applied to protect active agents, as the latter cannot move autonomously when encrypted.

The solution we advocate tries to combine the benefits of the two approaches we just discussed by introducing new primitives that provide for (i) subjective access control by ways of co-capabilities and (ii) data encryption to preserve secrecy of data while allowing agents to move autonomously.

We develop our approach for the calculus NBA of [7], a calculus of (boxed) ambients based on two ideas: direct, named communication across parent-child boundaries, and dynamic learning of incoming ambients' names. An NBA's ambient can be thought of as owning two channels, one for local, intra-ambient interactions, and one for hierarchical 'upward' communications. For instance $(x)^n.P \mid n[\,\langle m\rangle^{\uparrow}.Q\,]$ reduces directly to $P\{x := m\} \mid n[\,Q\,]$, and symmetrically with the roles of input and output swapped. Moreover, co-capabilities are binders, so that $a[\ \mathsf{in}\langle b,k\rangle.P\ ] \mid b[\ \overline{\mathsf{in}}(x,k).Q\ ]$ reduces to $b[\ a[P] \mid Q\{x := a\}\ \ ]$, and similarly for the out. This means that Q inside b has learned the name of the incoming agent a. Observe that k acts as a password (as in [16]), that must be matched for the action to take place. Actually, name binding and password checking work in a way at all analogous to the exchange of names and credential which occur when registering for a networked service.

Following the intuitions highlighted above, on top of communication mechanisms *à la* NBA, we introduce a specific primitive to let an ambient "seal" itself:

$$n[\,\mathsf{seal}\ k.P \mid Q\,] \ \longrightarrow\ n\{\!|\,P \mid Q\,|\!\}_k$$

By exercising the capability $\mathsf{seal}\ k$ in one of its internal threads, the ambient n blocks all its interactions with the outside, and encrypts all messages (to be exchanged either locally or across boundaries) present in n, included those contained in the thread Q. The flexibility of this model comes from the fact that a "sealed" ambient $n\{\!|\,P \mid Q\,|\!\}_k$ is still (partially) active: in particular, it may still freely move in the network, revealing its name), as well as performing limited forms of local synchornizazion. Instead, its exchanges are blocked and all data encrypted, and so remain until the ambient reaches a computational environment that knows the sealing key k. The mechanism to unseal a sealed ambient is associated to movements and co-capabilities containing keys as, e.g., in

$$n\{\!\!\{\, \mathsf{in}\ m.P \mid Q\,\}\!\!\}_k \mid m\{\,\overline{\mathsf{in}}\ \{x\}_k.R \mid S\,\} \longrightarrow m\{\,n[P \mid Q] \mid R\{x := n\} \mid S\,\}$$

The resulting model can, in some respects, be viewed as a symmetric crypto-system, with encryption associated with a sealing capability that secures all data inside an ambient, and decryption associated with the dual operation of unsealing performed at ambient boundaries.

2 Sealed Ambients

The syntax of the SBA calculus, in Table 1, is a proper extension of the syntax of Boxed Ambients [5], with movement co-capabilities and new 'sealing' primitives.

Table 1 Syntax

Expressions

M,N	$::=$	$k-q$	names
	$\mid$	$x-z$	variables
	$\mid$	$\mathsf{in}\ M$	enter M
	$\mid$	$\mathsf{out}\ M$	exit M
	$\mid$	$\overline{\mathsf{in}}$	let enter
	$\mid$	$\overline{\mathsf{out}}$	let exit
	$\mid$	$M.M$	path

Locations

η	$::=$	a	child
	$\mid$	$\uparrow$	parent
	$\mid$	$\star$	local

Prefixes

π	$::=$	M	path
	$\mid$	$(x_1,\ldots,x_k)^\eta$	input
	$\mid$	$\langle M_1,\ldots,M_k\rangle^\eta$	output
	$\mid$	$\overline{\mathsf{in}}\ \{x\}_M$	unseal entering ambient
	$\mid$	$\overline{\mathsf{out}}\ \{x\}_M$	unseal exiting ambient
	$\mid$	$\mathsf{seal}\ M$	sealing

Processes

P	$::=$	$\mathbf{0}$	stop
	$\mid$	$\pi.P$	prefixing
	$\mid$	$(\nu n)P$	restriction
	$\mid$	$P \mid P$	composition
	$\mid$	$!P$	replication
	$\mid$	$M[P]$	ambient
	$\mid$	$M\{\!\!\{P\}\!\!\}_N$	sealed ambient

Names and variables range over two disjoint sets. We use $a-d$ to denote names and variables, when the distinction is immaterial. Messages are formed as usual over names and (paths of) capabilities. The operators of inactivity, composition, restriction and replication are inherited from the π-calculus [17]. The process forms $(x_1,\ldots,x_k)^\eta.P$, $\langle M_1,\ldots,M_k\rangle^\eta.P$ and $M[P]$ denote directed (synchronous) input/output as in the calculus of Boxed Ambients [5], and ambients as in the Ambient Calculus [10]. In addition SBA provides a new construct for the formation of *sealed* ambients, noted $M\{\!\!\{P\}\!\!\}_N$, where M is the name and N is the sealing key. Three new prefix forms provide for the operations of unsealing, $\overline{\mathsf{in}}\ \{x\}_k.P$, $\overline{\mathsf{out}}\ \{x\}_k.P$, and encryption $\mathsf{seal}\ k.P$.

We use the usual notation conventions. Parallel composition has the lowest precedence among the operators. The process $\pi_1.\pi_2.P$ is read as $\pi_1.(\pi_2.P)$, while $\langle\tilde{M}\rangle^\eta, (\tilde{x})$ and $(\nu\tilde{n})$ stand for $\langle M_1,\ldots,M_k\rangle^\eta, (x_1,\ldots,x_k)^\eta$, $(\nu\ n_1,\ldots,n_k)$, respectively. We omit trailing and isolated dead processes, writing π for $\pi.\mathbf{0}$, $\langle\tilde{M}\rangle$ for $\langle\tilde{M}\rangle.\mathbf{0}$, and $n[\ ]$ for $n[\mathbf{0}]$. The superscript $\star$ denoting local communication, is omitted. The operators $(\nu n)P$, $\overline{\mathsf{in}}\ \{x\}_a.P$, $\overline{\mathsf{out}}\ \{x\}_a.P$, and $(\tilde{x})^\eta.P$ act as binders for the name n, and the variables x and $\tilde{x}$, respectively. The sets of *free names* and *free variables* of P, $fn(P)$ and $fv(P)$, are defined accordingly. A process is closed if it has no free variables (though it may have free names). In addition, we write $M\{\,P\,\}$ for $M\{\!\!\{P\}\!\!\}_N$ or $M[P]$ when the distinction may safely be disregarded.

2.1 Reduction

The operational semantics is defined as usual in terms of reduction and structural congruence. The definition of structural congruence is standard [10]. The basic idea behind the reduction relation is that ambients can be in two states, either *sealed* or *unsealed*. An ambient may be sealed at its formation, or become sealed as a result of one of its enclosed threads exercising a capability. When sealed, an ambient may move but not exchange any value, either locally or with the context. Instead, an unsealed ambient is fully operational and may move as well as communicate. The two states for reductions are formalized by defining the reduction relation in terms of two, inter-dependent relations, formalized in Tables 2 and 3.

The relation $\dashrightarrow$ (referred to as *silent reduction*) formalizes the semantics of mobility and sealing. In particular, the two rules (*enter*) and (*exit*) allow any ambient, sealed or unsealed, to traverse any other ambient (sealed or unsealed): the move requires the target ambient to co-operate by offering a co-capability. The two rules (*K-enter*) and (*K-exit*) provide an alternative mechanism for mobility, akin to that studied in [7]. As in *loc. cit.*, the incoming ambient is authenticated by a test on the sealing key k, and then its name registered by binding its name to the variable x. In addition, in SBA, the authentication mechanism has also the effect removing the seal on the incoming ambient, so as to enable it to interact with the accepting context. The rule *(seal)* shows the effect of sealing: the capability $\mathsf{seal}\ k$ instructs a process to seal its enclosing ambient under a key k.

Silent reductions may occur within any context, except under prefixes. On the other hand, the reductions for communication, formalized by the relation $\longrightarrow$, may only occur within unsealed ambients. The rules for communication are exactly as in previous versions of Boxed Ambients [11,7]: as we noted, only unsealed ambients can hold exchanges with their enclosing context. In addition, as we said above, no exchange (local or across boundaries) may occur within sealed ambients: this is a consequence of how the notion of evaluation context is given in Table 3. The rule (*struct*) is standard, while the two rules (*silent*) and (*ambient*) guarantee that the two relations are linked properly.

Remarks. Notice that syntax and operational semantics are so defined as to guarantee that ambients cannot be sealed more than once. This is only for convenience, and ease of presentation. Note, further, that an alternative presentation of the calculus could be given by separating the sealing primitives from those for mobility. Specifically, one could introduce an explicit unsealing prefix such as $\mathsf{unseal}\{x\}_k.P$, and define its semantics by the reduction $\mathsf{unseal}\{x\}_k.P \mid n\{\!\!\{ Q \}\!\!\}_k \longrightarrow P\{x := n\} \mid n[Q]$. This rule, together with rules (*enter*) and (*exit*) would implement an equivalent (albeit not atomic) unsealing mechanism. However, our present solution appears to model faithfully the current practice in distributed and mobile systems, where the protocols for agent authentication and certification take place at domain (i.e. ambient) boundaries rather than after such boundaries have been crossed.

2.2 Examples

The kind of secure communication expected of the exchange of messages in (3) can now be achieved by coding the protocol as follows:

$$(\nu n)a[\ p[\mathsf{seal}\ n.\mathsf{out}\ a.\mathsf{in}\ b.\langle m\rangle^{\uparrow}\,]\ ] \mid \overline{\mathsf{out}}\ \mid b[\overline{\mathsf{in}}\ \{x\}_n.(y)^x.P \mid Q]$$

The public ambient p seals itself with the private key s, shared by the sender and the intended receiver, moves over the network towards its destination, gets unsealed in the act of entering it, and becomes then ready to deliver its message. Like in spi-calculus specification of the protocol, it is the sealing key that is private, while the name of the ambient may be left public.

Table 2 Silent Reduction

Silent Reduction Context $\mathbf{S} ::= - \mid (\nu n)\mathbf{S} \mid P|\mathbf{S} \mid n[\mathbf{S}] \mid n\{\!|\mathbf{S}|\!\}_k$

Mobility (I)

$$(\textit{enter}) \quad n\{\,\mathsf{in}\ m.P \mid Q\,\} \mid m\{\,\overline{\mathsf{in}}\ .R \mid S\,\} \dashrightarrow m\{\,n\{\,P \mid Q\,\} \mid R \mid S\,\}$$

$$(\textit{exit}) \quad m\{\,n\{\,\mathsf{out}\ m.P \mid Q\,\} \mid R\,\} \mid \overline{\mathsf{out}}\ .S \dashrightarrow n\{\,P \mid Q\,\} \mid m\{\,R\,\} \mid S$$

Mobility (II)

$$(\textit{K-enter}) \quad n\{\!|\,\mathsf{in}\ m.P \mid Q\,|\!\}_k \mid m\{\,\overline{\mathsf{in}}\ \{x\}_k.R \mid S\,\} \dashrightarrow m\{\,n[\,P \mid Q\,] \mid R\{x := n\} \mid S\,\}$$

$$(\textit{K-exit}) \quad m\{\,P \mid n\{\!|\,\mathsf{out}\ m.Q \mid R\,|\!\}_k\,\} \mid \overline{\mathsf{out}}\ \{x\}_k.S \dashrightarrow m\{\,P\,\} \mid n[\,Q \mid R\,] \mid S\{x := n\}$$

$$(\textit{seal}) \quad n[\,\mathsf{seal}\ k.P \mid Q\,] \dashrightarrow n\{\!|\,P \mid Q\,|\!\}_k$$

Structural Rules

$$(\textit{struct}) \quad P \equiv Q, Q \dashrightarrow R,\ R \equiv S \ \Rightarrow\ P \dashrightarrow S$$

$$(\textit{context}) \quad P \dashrightarrow Q \ \Rightarrow\ \mathbf{S}\{P\} \dashrightarrow \mathbf{S}\{Q\}$$

Table 3 Reduction

Reduction Context $\mathbf{E} ::= - \mid (\nu n)\mathbf{E} \mid P|\mathbf{E} \mid n[\mathbf{E}]$

Communication

$$(\textit{local}) \quad (\tilde{x})P \mid \langle \tilde{M} \rangle Q \longrightarrow P\{\tilde{x} := \tilde{M}\} \mid Q$$

$$(\textit{input } n) \quad (\tilde{x})^n P \mid n[\,\langle \tilde{M} \rangle^{\uparrow} Q \mid R\,] \longrightarrow P\{\tilde{x} := \tilde{M}\} \mid n[\,Q \mid R\,]$$

$$(\textit{output } n) \quad \langle \tilde{M} \rangle^n P \mid n[\,(\tilde{x})^{\uparrow} Q \mid R\,] \longrightarrow P \mid n[\,Q\{\tilde{x} := \tilde{M}\} \mid R\,]$$

Structural Rules

$$(\textit{silent}) \quad P \dashrightarrow Q \ \Rightarrow\ P \longrightarrow Q$$

$$(\textit{ambient}) \quad P \longrightarrow Q \ \Rightarrow\ n[P] \dashrightarrow n[Q]$$

$$(\textit{struct}) \quad P \equiv Q, Q \longrightarrow R,\ R \equiv S \ \Rightarrow\ P \longrightarrow S$$

$$(\textit{context}) \quad P \longrightarrow Q \ \Rightarrow\ \mathbf{E}\{P\} \longrightarrow \mathbf{E}\{Q\}$$

Incidentally, this formulation of the message exchange fixes a minor flaw in the protocols we discussed in §1 above. Namely, in the configuration $b[\,\mathsf{open}\ n.(x).Q \mid Q' \mid n[\,\overline{\mathsf{open}}\ n.\langle m \rangle\,]\,]$, as the opening of n and the delivery of its message are distinct steps, when multiple threads are present inside b, there is no guarantee that m will be received by the intended process. In particular, m could end up in Q', even when it did not actually know the secret name n. Such behaviour is however inherent to the communication model of MA: the problem fades away with the primitives for hierarchical communication of the present calculus.

As a more realistic example, consider the case of an agent in search of vendors of a particular item over the network. The agent originates at a user site u, visits a collection $\mathsf{s_i}$ of network sites and reports the names of the sites that provide a specific item it. To protect the agent on the move across the network, we use the SBA primitives as follows. Let k be a sealing key shared between the user u and the sites $\mathsf{s_i}$. The user can be represented as the process $\mathsf{u}[\,(\nu a)a[P \mid R] \mid Q\,]$, where a is an agent with two threads: a router R, which controls movements, and a communicator P, which interacts with the visited sites. We use two locks l and r to synchronize the two threads within a:

$$\begin{aligned} a[\,(\nu l, r)(\mathsf{synch}(l) \ &\mid\ !\,\overline{\mathsf{synch}}(l).\mathsf{seal}\ k.(\mathsf{synch}(r) \mid \langle \mathsf{it} \rangle^{\uparrow}.(x,y)^{\uparrow}([x = \mathsf{it}]\langle y \rangle \mid \mathsf{synch}(l))) \\ &\mid \overline{\mathsf{synch}}(r).\mathsf{route}(\mathsf{u}, \mathsf{s}_1).\overline{\mathsf{synch}}(r).\mathsf{route}(\mathsf{s}_1, \mathsf{s}_2).\overline{\mathsf{synch}}(r).\mathsf{route}(\mathsf{s}_2, \mathsf{u})\,]\,. \end{aligned}$$

The first thread is a loop that, when activated, seals the agents under the key k, then activates the router and waits for a to be routed to the destination sites. Once there, it collects the name of the vendor, if this contains the desired item. The router thread, in turn, ships a across the network to visit the sites, in this case s_1 and s_2. However, before moving outside u, or any of the s_i's, it waits for the sibiling thread to seal the agent using k. The reduction semantics guarantees that, whenever the ambient a is not inside any site which knows k, all data in a are sealed, hence kept secret.

To synchronize with each other, the router and the communicator use the following process forms: $\mathsf{synch}(n) \triangleq n[n\{\!|\,\mathsf{out}\ n\,|\!\}_n]$, and $\overline{\mathsf{synch}}(n) \triangleq \overline{\mathsf{out}}\ \{\_\}_n$, for each name n. Interestingly, the local synchronization between the two threads, which does not rely on exchanges of messages, is available even though the ambient is sealed. The definition also uses the following simple form of testing for equality of names:

$$[a=b]P \triangleq (\nu c)\ (c[\langle\,\rangle^a \mid b[(\,)^\uparrow.\langle\,\rangle^\uparrow]\mid (\,)^b.\langle\,\rangle^\uparrow]\mid (\,)^c.P) \qquad (c \notin fn(P))$$

The process $[a=b]P$ behaves like P if a and b are the same name, and is stuck otherwise. Finally, each of the visiting sites can be coded as follows:

$$\mathsf{s_i}[\overline{\mathsf{in}}\ \{z\}_k.(x)^z.\langle f_i(x),\mathsf{s_i}\rangle^z \mid \ldots]$$

When the agent a enters $\mathsf{s_i}$ it gets unsealed, so that it may hold exchanges with the site. Here the function f_i represents a calculation calculation performed by the site to find out if it contains the item x: the result of such a calculation is x if $\mathsf{s_i}$ contains x among its items on sale, and some other (different) value otherwise.

3 A Type System

The type system separates trusted and untrusted data and code while allowing safe interactions with untrusted sites. In particular, a distinct type Un is used to type processes for which we cannot make any assumption on structure and/or behavior. Correspondingly, we assign a 'default' type Public to data that comes from untyped processes, and we handle such values very carefully, as we have no knowledge of what a particular piece of data they are. The structure of types is defined by the following productions:

$$\begin{array}{llcll}
\textit{Expression Types} & W & ::= & \mathsf{Amb}[E] & \text{ambient} \\
 & & \mid & \mathsf{Key}[E] & \text{key} \\
 & & \mid & \mathsf{Public} & \text{unknown} \\
\textit{Exchange Types} & E,F & ::= & \mathsf{shh} & \text{no exchange} \\
 & & \mid & (W_1,\ldots,W_k) & \text{tuple exchange} \\
\textit{Process Types} & T & ::= & [E,F] & \text{process} \\
 & & \mid & \mathsf{Un} & \text{untrusted process}
\end{array}$$

As we said, the type Public is assigned to expressions that may be exchanged with untrusted processes. Among such expressions, we include the movement (co-)capabilities, so as to enhance the flexibility of typing: there is no negative effect on safety (or security) in this choice, as the interaction among trusted components is enabled by the possession of shared keys, which are secret and hence protected from the untrusted components. The type $\mathsf{Key}[E]$ is the type of sealing keys: a key with this type may only be used to seal ambients of type $\mathsf{Amb}[E]$. The latter, in turn, is the type of all the ambients whose upward exchanges (if any) have type E. Notice that only ambients (not generic expressions) can be sealed; furthermore, the types of keys are considered 'secret' and hence protected.

As for process types, $[E,F]$ is the type of all processes that can be enclosed in ambients of type $\mathsf{Amb}[F]$, with E and F denoting the local and upward exchanges of the processes in question. Un is the type of the untrusted processes.

In order to provide the intended privacy guarantees, the types of trusted and untrusted data and processes are kept separate (there is no subsumption rule, nor any common super-type). Nevertheless, the typing rules for processes allow non trivial forms of interactions between trusted and untrusted processes. Specifically, ambients have full freedom of mobility, in that trusted ambients may traverse untrusted ones and vice versa. Instead, unsealing is only allowed between trusted ambients. As for communication, the following policy is adopted: (i) local exchanges between trusted and untrusted processes are disallowed, (ii) trusted ambients may communicate Public values to untrusted ones, and (iii) untrusted ambients may communicate Public values to trusted ones. We proceed with the description of the typing rules, which are collectively reported in Tables 3 and 4.

3.1 Typing Rules

Every (co-)capability is assigned type Public; given that, the rule (PREFIX), allows trusted ambients to traverse untrusted ones and vice versa, without breaking the soundness of the type system. Note that ill-formed (paths of) capabilities, such as $a.b$ and $\mathsf{in}\ (a.b)$ do type check in this system when a,b are Public. This is necessary to allow full flexibility in the typing of the opponent: on the other hand, as we shall prove, the type system does provide the expected guarantees of secrecy, and of safety for any process exchange via communication.

Each process form has two associated typing rules, depending on whether the process in question is to be considered trusted (Table 4) or deemed untrusted (Table 5): in the latter case, it could be an attacker or a trusted process tainted by an interaction with an untrusted component via its public names. For prefixes the two cases can be accounted for by a single rule, (PREFIX), where T stands for either $[E,F]$ or Un. For ambients we have four rules: (AMB SEAL) in Table 3 assigns a type to ambients formed with the 'right' key N, and enclosing a process P with the expected exchanges; (AMB) is standard, while the rules (UNTRUSTED AMB/AMB SEAL), in Table 4 are used to type untrusted, possibly ill-formed, ambients. In addition, observe that a trusted (sealed) ambient $a\{P\}$ may be typed with type Un; this is perfectly correct and allows a trusted (sealed) ambient to traverse untrusted sites.

The same rationale applies to the prefix constructors for sealing and unsealing, as well as for local and upward communication. Three typing rules handle the case of input (output) from a sub-ambient M. To enhance the flexibility of the type system, we allow untrusted and trusted process to exchange values, as long as these have type Public – cf. rules (UNTRUSTED INPUT/OUTPUT M) in Table 4. In addition, in these rules we do not require that the arity of the downward communication matches that of the ambient M. This again allows a flexible typing of opponent processes.

The type system satisfies the expected soundness property:

Proposition 1 (Subject Reduction). *If $\Gamma \vdash P : T$ and $P \longrightarrow Q$, then $\Gamma \vdash Q : T$.*

In addition, the following proposition, corresponding to the usual 'typability lemma,' states that any process P can be typed with type Un assuming that its names and variables are of type Public. In other words, the type system does not impose any constraint on the structure of the opponent: only that it initially does not know any secret.

Proposition 2 (Typability). *For all P, if $fn(P) = \{a_1,\ldots,a_n\}$ and $fv(P) = \{x_1,\ldots,x_m\}$, then* $a_1 : \mathsf{Public},\ldots,a_n : \mathsf{Public}, x_1 : \mathsf{Public},\ldots,x_m : \mathsf{Public} \vdash P : \mathsf{Un}$.

Table 4 Typing Rules: Trusted Processes

(EMPTY)
$$\frac{}{\emptyset \vdash \diamond}$$

(ENV x)
$$\frac{\Gamma \vdash \diamond \quad x \notin Dom(\Gamma)}{\Gamma, x : W \vdash \diamond}$$

(PROJECTION)
$$\frac{\Gamma \vdash \diamond \quad \Gamma(M) = W}{\Gamma \vdash M : W}$$

(PATH)
$$\frac{\Gamma \vdash M_1 : \mathsf{Public} \quad \Gamma \vdash M_2 : \mathsf{Public}}{\Gamma \vdash M_1.M_2 : \mathsf{Public}}$$

(CO-IN)
$$\frac{\Gamma \vdash \diamond}{\Gamma \vdash \overline{\mathsf{in}} : \mathsf{Public}}$$

(CO-OUT)
$$\frac{\Gamma \vdash \diamond}{\Gamma \vdash \overline{\mathsf{out}} : \mathsf{Public}}$$

(IN M)
$$\frac{\Gamma \vdash M : W \quad W \in \{\mathsf{Amb}[E], \mathsf{Public}\}}{\Gamma \vdash \mathsf{in}\ M : \mathsf{Public}}$$

(OUT M)
$$\frac{\Gamma \vdash M : W \quad W \in \{\mathsf{Amb}[E], \mathsf{Public}\}}{\Gamma \vdash \mathsf{out}\ M : \mathsf{Public}}$$

(PREFIX)
$$\frac{\Gamma \vdash M : \mathsf{Public} \quad \Gamma \vdash P : T}{\Gamma \vdash M.P : T}$$

(SEAL)
$$\frac{\Gamma \vdash M : \mathsf{Key}[E] \quad \Gamma \vdash P : [F, E]}{\Gamma \vdash \mathsf{seal}\ M.P : [F, E]}$$

(AMB)
$$\frac{\Gamma \vdash M : \mathsf{Amb}[E] \quad \Gamma \vdash P : [F, E]}{\Gamma \vdash M[P] : T}$$

(AMB SEAL)
$$\frac{\Gamma \vdash N : \mathsf{Key}[E] \quad \Gamma \vdash M : \mathsf{Amb}[E] \quad \Gamma \vdash P : [F, E]}{\Gamma \vdash M\{\!|P|\!\}_N : T}$$

(CO-IN KEY)
$$\frac{\Gamma \vdash M : \mathsf{Key}[E] \quad \Gamma, x{:}\mathsf{Amb}[E] \vdash P : [G, H]}{\Gamma \vdash \overline{\mathsf{in}}\ \{x\}_M.P : [G, H]}$$

(CO-OUT KEY)
$$\frac{\Gamma \vdash M : \mathsf{Key}[E] \quad \Gamma, x{:}\mathsf{Amb}[E] \vdash P : [G, H]}{\Gamma \vdash \overline{\mathsf{out}}\ \{x\}_M.P : [G, H]}$$

(DEAD)
$$\frac{\Gamma \vdash \diamond}{\Gamma \vdash \mathbf{0} : T}$$

(PAR)
$$\frac{\Gamma \vdash P : T \quad \Gamma \vdash Q : T}{\Gamma \vdash P \mid Q : T}$$

(NEW)
$$\frac{\Gamma, n : W \vdash P : T}{\Gamma \vdash (\nu n)P : T}$$

(REPL)
$$\frac{\Gamma \vdash P : T}{\Gamma \vdash !P : T}$$

(LOCAL INPUT)
$$\frac{\Gamma, x_1 : W_1, \ldots, x_k : W_k \vdash P : [(W_1, \ldots, W_k), E]}{\Gamma \vdash (x_1, \ldots, x_k).P : [(W_1, \ldots, W_k), E]}$$

(LOCAL OUTPUT) $i = 1, ..., k$
$$\frac{\Gamma \vdash M_i : W_i \quad \Gamma \vdash P : [\tilde{W}, E]}{\Gamma \vdash \langle \tilde{M} \rangle.P : [\tilde{W}, E]}$$

(INPUT $\uparrow$)
$$\frac{\Gamma, x_1 : W_1, \ldots, x_k : W_k \vdash P : [E, (W_1, \ldots, W_k)]}{\Gamma \vdash (x_1, \ldots, x_k)^{\uparrow}.P : [E, (W_1, \ldots, W_k)]}$$

(OUTPUT $\uparrow$) $i = 1, ..., k$
$$\frac{\Gamma \vdash M_i : W_i \quad \Gamma \vdash P : [E, (W_1, \ldots, W_k)]}{\Gamma \vdash \langle M_1, \ldots, M_k \rangle^{\uparrow}.P : [E, (W_1, \ldots, W_k)]}$$

(INPUT M)
$$\frac{\Gamma \vdash M : \mathsf{Amb}[\tilde{W}] \quad \Gamma, \tilde{x} : \tilde{W} \vdash P : [E, F]}{\Gamma \vdash (\tilde{x})^M.P : [E, F]}$$

(OUTPUT M AMB)
$$\frac{\Gamma \vdash N : \mathsf{Amb}[\tilde{W}] \quad \Gamma \vdash \tilde{M} : \tilde{W} \quad \Gamma \vdash P : [E, F]}{\Gamma \vdash \langle \tilde{M} \rangle^N.P : [E, F]}$$

Table 5 Typing Rules: Untrusted Processes

(UNTRUSTED AMB)

$$\frac{\Gamma \vdash M : \mathsf{Public} \quad \Gamma \vdash P : \mathsf{Un}}{\Gamma \vdash M[P] : T}$$

(UNTRUSTED AMB SEAL)

$$\frac{\Gamma \vdash N : \mathsf{Public} \quad \Gamma \vdash M : \mathsf{Public} \quad \Gamma \vdash P : \mathsf{Un}}{\Gamma \vdash M\{\!|P|\!\}_N : T}$$

(UNTRUSTED SEAL)

$$\frac{\Gamma \vdash M : \mathsf{Public} \quad \Gamma \vdash P : \mathsf{Un}}{\Gamma \vdash \mathsf{seal}\ M.P : \mathsf{Un}}$$

(UNTRUSTED CO-IN)

$$\frac{\Gamma \vdash M : \mathsf{Public} \quad \Gamma, x{:}\mathsf{Public} \vdash P : \mathsf{Un}}{\Gamma \vdash \overline{\mathsf{in}}\ \{x\}_M.P : \mathsf{Un}}$$

(UNTRUSTED CO-OUT)

$$\frac{\Gamma \vdash M : \mathsf{Public} \quad \Gamma, x{:}\mathsf{Public} \vdash P : \mathsf{Un}}{\Gamma \vdash \overline{\mathsf{out}}\ \{x\}_M.P : \mathsf{Un}}$$

(UNTRUSTED LOCAL INPUT)

$$\frac{\Gamma, x_1 : \mathsf{Public}, \ldots x_k : \mathsf{Public} \vdash P : \mathsf{Un}}{\Gamma \vdash (x_1, \ldots, x_k).P : \mathsf{Un}}$$

(UNTRUSTED LOCAL OUTPUT)

$$\frac{\Gamma \vdash M_i : \mathsf{Public} \ \ i = 1, ..., k \quad \Gamma \vdash P : \mathsf{Un}}{\Gamma \vdash \langle M_1, \ldots, M_k \rangle.P : \mathsf{Un}}$$

(UNTRUSTED INPUT $\uparrow$)

$$\frac{\Gamma, x_1 : \mathsf{Public}, \ldots x_k : \mathsf{Public} \vdash P : \mathsf{Un}}{\Gamma \vdash (x_1, \ldots, x_k)^{\uparrow}.P : \mathsf{Un}}$$

(UNTRUSTED OUTPUT $\uparrow$)

$$\frac{\Gamma \vdash M_i : \mathsf{Public} \quad i = 1, ..., k \quad \Gamma \vdash P : \mathsf{Un}}{\Gamma \vdash \langle M_1, \ldots, M_k \rangle^{\uparrow}.P : \mathsf{Un}}$$

(INPUT M UNTRUSTED)

$$\frac{\Gamma \vdash M : \mathsf{Public} \quad \Gamma, x_1 : \mathsf{Public}, \ldots x_k : \mathsf{Public} \vdash P : T}{\Gamma \vdash (x_1, \ldots, x_k)^M.P : T}$$

(UNTRUSTED INPUT M)

$$\frac{\Gamma \vdash M : \mathsf{Amb}[\mathsf{Public}_1, \ldots, \mathsf{Public}_n] \quad \Gamma, x_1 : \mathsf{Public}, \ldots x_k : \mathsf{Public} \vdash P : \mathsf{Un}}{\Gamma \vdash (x_1, \ldots, x_k)^M.P : \mathsf{Un}}$$

(OUTPUT M UNTRUSTED)

$$\frac{\Gamma \vdash M : \mathsf{Public} \quad \Gamma \vdash M_i : \mathsf{Public} \quad i = 1, ..., k \quad \Gamma \vdash P : T}{\Gamma \vdash \langle M_1, \ldots, M_k \rangle^M.P : T}$$

(UNTRUSTED OUTPUT M)

$$\frac{\Gamma \vdash M : \mathsf{Amb}[\mathsf{Public}_1, \ldots, \mathsf{Public}_n] \quad \Gamma \vdash M_i : \mathsf{Public} \ \ i = 1, .., k \quad \Gamma \vdash P : \mathsf{Un}}{\Gamma \vdash \langle M_1, \ldots, M_k \rangle^M.P : \mathsf{Un}}$$

4 A Secrecy Theorem

We refer to a rather standard notion of secrecy in the literature of security protocols, namely: *a process preserves the secrecy of a piece of data M if it does not publish M, or anything that would permit the computation of M*. The formal definition is inspired by [2] which we adapt to our framework, by representing an attacker as a closed, but otherwise arbitrary, context $\mathbf{A}()$. This leaves full power to an attacker, which can either take the role of an hostile context (or host) enclosing a trusted process, as in $a[\,Q \mid ()\,]$, or the role of a malicious agent mounting an attack to a remote host, as in $a[\,\mathsf{in}\ p.\mathsf{in}\ q.Q \mid Q'\,] \mid ()$. In addition, we characterize the initial knowledge of the attacker in terms of the names, the keys and the capabilities initially known to it. Perhaps interestingly, the knowledge of capabilities is important here, since by exercising (a path of) capabilities an adversary may approach an agent and interact with it, even without knowing its name. As an example, if we take the process $b[\,\overline{\mathsf{in}}\ \{x\}_k.\langle a\rangle^x\,]$, an opponent may have access to the value a even without knowing the name b: knowing the capability '$\mathsf{in}\ b$' and the key $'k'$ is enough.

We denote with $fc(P)$ the set of capabilities formed over the free names of P: the inductive definition of this set is straightforward.

Definition 1 (S-adversary). *Let S be a finite set of names and capabilities. The closed context $\mathbf{A}()$ is an S-adversary if $fn(\mathbf{A}()) \cup fc(\mathbf{A}()) \subseteq S$.*

Next, we define what it means to preserve a secret: since capabilities are public, the definition of secrecy only applies to names. Let $\Longrightarrow$ be the reflexive and transitive closure of the reduction relation $\longrightarrow$.

Definition 2 (Revealing Names, Preserving their Secrecy). *Let P be a process, n a name and S a finite set of names and capabilities. P* may reveal *n* to *S iff there exists an S-adversary $\mathbf{A}()$, with $\mathbf{A}(P)$ is closed, and a name $c \in S$ such that $\mathbf{A}(P) \Longrightarrow \mathbf{C}(c[\,\langle n\rangle^{\uparrow} \mid Q\,])$, for some context $\mathbf{C}()$ and process Q, with c not bound by $\mathbf{C}()$. Dually, P* preserves the secrecy of *n* from *S iff it does not reveal m to S.*

The definition extends readily to private names as follows (cf. [9]): $(\nu n)P$ *may reveal n to S* if and only if there is a name $P\{n := m\}$ may reveals m to S, with $m \notin S \cup fn(P)$.

Notice that an adversary may dynamically acquire new names and new capabilities by either creating its own fresh names and, more importantly, by receiving names over public channels and by unsealing ambients sealed with a key it knows (thus learning the ambient's name). As an example, take $S = \{c\}$, and consider the process $P = c[\,\langle a\rangle^{\uparrow}\,] \mid a[\,\langle k\rangle^{\uparrow}\,]$. P does not preserve the secrecy of k from S, even though S does not include a. In fact, one can take the S-adversary $\mathbf{A}() = (x)^c.(y)^x.c[\,\langle y\rangle^{\uparrow}\,] \mid ()$, and note that $\mathbf{A}(P) \Longrightarrow c[\,\langle k\rangle^{\uparrow}\,] \mid c[\,] \mid a[\,]$.

The secrecy theorem, below, states that a well-typed process P does not leak its secrets to any adversary that initially knows all the public names in P and has the capabilities to move in and out any ambient of P (included its secret ambients).

Note that, by definition, an adversary has access to the anonymous top level channel of any process. Thus a (trusted) agent that actually allows interactions with adversaries must be such that its top level processes have type Un. Let indeed $\mathbf{A}()$ be an adversary context built with public names and capabilities contained in a process P; by Typability Lemma 2, in order for P to interact with the opponent, i.e. to fill the hole in the context $\mathbf{A}()$, it must have type Un (remind that trusted processes, in particular trusted ambients, may be safely typed with type Un). This explains the hypothesis of the following secrecy theorem. There are obviously other safe processes, in particular those that can be typed with type $[E,F]$, but we do not consider them since their typing implies that they can be put only within trusted contexts.

Theorem 1 (Secrecy). *Let P be a process such that $\Gamma \vdash P : Un$ and $\Gamma \vdash s : W$ with $W \neq \mathsf{Public}$. Let $S = \{a \mid \Gamma \vdash a : \mathsf{Public}\} \cup \{\mathsf{in}\ a, \mathsf{out}\ a \mid a \in Dom(\Gamma)\}$. Then P preserves the secrecy of s from S.*

5 Encoding of the Spi Calculus

As a further example, we give an encoding the following fragment of the asynchronous spi-calculus, where we disregard the construct for pairs, natural numbers and matching.

$$\begin{array}{lrcl} \textit{Expressions} & M,N & ::= & n \mid x \mid \{M_1,\dots,M_n\}_N \\ \textit{Processes} & P,Q & ::= & \mathbf{0} \mid \overline{M}\langle N_1,\dots,N_n\rangle \mid M(x_1,\dots,x_n)P \mid P|Q \\ & & \Big| & (\nu n)P \mid \mathit{case}\ M\ \mathit{of}\ \{x_1,\dots,x_n\}_N\ \mathit{in}\ P \end{array}$$

The operational semantics of this fragment is standard (cf. [3]): in particular, the decryption of an encrypted packed is governed by the following reduction:

$$\mathit{case}\ \{M_1,\dots,M_n\}_k\ \mathit{of}\ \{x_1,\dots,x_n\}_k\ \mathit{in}\ P \longrightarrow P\{x_i := M_i\}$$

The basic idea of the encoding is to represent an encrypted message with a sealed ambient that contains that message: communicating the encrypted messages is then accounted for by communicating the name of the corresponding ambient.

Table 6 Encoding of the spi calculus

$$\begin{array}{l} \langle\!\langle a \rangle\!\rangle_p = a \\ \langle\!\langle \{M_1,\dots,M_n\}_k \rangle\!\rangle_p = p \\[1ex] [\![a]\!]_p = \mathbf{0} \\ [\![\{M_1,\dots,M_n\}_k]\!]_p = (\nu q_1,\dots,q_n)(\Pi_{i=1}^n [\![M_i]\!]_{q_i} \mid \\ \qquad\qquad p[\,(x)^{\uparrow}.\mathsf{seal}\ k.\mathsf{in}\ x.\langle \langle\!\langle M_1 \rangle\!\rangle_{q_1},\dots,\langle\!\langle M_n \rangle\!\rangle_{q_n}\rangle^{\uparrow}\,]) \\[1ex] [\![\mathbf{0}]\!] = \mathbf{0},\ [\![(\nu n)P]\!] = (\nu n)[\![P]\!],\ [\![P \mid Q]\!] = [\![P]\!] \mid [\![Q]\!] \\ [\![\overline{b}\langle M_1,\dots,M_n\rangle]\!] = (\nu q_1,\dots,q_n)(\Pi_{i=1}^n [\![M_i]\!]_{q_i} \mid b[\,\langle \langle\!\langle M_1 \rangle\!\rangle_{q_1},\dots,\langle\!\langle M_n \rangle\!\rangle_{q_n}\rangle^{\uparrow}\,]) \\ [\![b(x_1,\dots,x_n)P]\!] = (x_1,\dots,x_n)^b.[\![P]\!] \\ [\![\mathit{case}\ M\ \mathit{of}\ \{x_1,\dots,x_n\}_k\ \mathit{in}\ P]\!] = (\nu p)([\![M]\!]_p \mid (\nu c)(\langle c\rangle^{\langle M\rangle_p} \mid \\ \qquad\qquad c[\,\overline{\mathsf{in}}\ \{y\}_k.(x_1,\dots,x_n)^y.\langle x_1,\dots,x_n\rangle^{\uparrow}\,] \mid (x_1,\dots,x_n)^c.P)) \end{array}$$

The formal definition is given in Table 5, in terms of three translation maps, with $[\![\cdot]\!]$ leading (and p a name):

$$\begin{array}{ll} \langle\!\langle \cdot \rangle\!\rangle_p & : \text{Expressions} \mapsto \text{Expressions} \\ [\![\cdot]\!]_p & : \text{Expressions} \mapsto \text{Processes} \\ [\![\cdot]\!] & : \text{Processes} \mapsto \text{Processes} \end{array}$$

$\langle\!\langle M \rangle\!\rangle_p$ returns the data M if M is a name or a variable; instead, if M is an encryption packet, it returns p, which is the name of (i.e. the pointer to) the ambient that stores the packet. Correspondingly, $[\![M]\!]_p$ stores M into an ambient named p, if M is an encrypted message, and returns the inactive process otherwise. More precisely, if M is a message encrypted under a key k, the ambient generated by $[\![M]\!]_p$ first waits to reads the name x of an ambient where it is expected to enter to be unsealed, then gets sealed with the key k to move into x, where it eventually delivers the data (after having been unsealed. Dually, decryption is obtained by letting the ambient p enter a temporary ambient c where p is unsealed with the key k.

To exemplify, with $P = \overline{a}\langle\{m\}_k\rangle \mid a(x).case\ x\ of\ \{y\}_k\ in\ P'$, we have:

$$\begin{array}{rl}
[\![P]\!] = & (\nu q)([\![\{M\}_k]\!]_q \mid a[\langle\langle\!\langle\{M\}_k\rangle\!\rangle_q\rangle^{\uparrow}]) \mid \\
& (x)^a(\nu p)([\![x]\!]_p \mid (\nu c)(\langle c\rangle^{\langle x\rangle_p} \mid c[\overline{\mathsf{in}}\ \{y\}_k.(x)^y.\langle x\rangle^{\uparrow}] \mid (y)^c.[\![P']\!])) \\
\equiv & (\nu q, q_1)([\![M]\!]_{q_1} \mid q[(x)^{\uparrow}.\mathsf{seal}\ k.\mathsf{in}\ x.\langle\langle\!\langle M\rangle\!\rangle_{q_1}\rangle^{\uparrow}] \mid a[\langle q\rangle^{\uparrow}]) \mid \\
& \qquad (x)^a.(\nu c)(\langle c\rangle^x \mid c[\overline{\mathsf{in}}\ \{y\}_k.(x)^y.\langle x\rangle^{\uparrow}] \mid (y)^c.[\![P']\!]) \\
\Longrightarrow & (\nu q, q_1, c)([\![M]\!]_{q_1} \mid a[\ ] \mid c[q[\ ]\ ] \mid [\![P']\!]\{y := \langle\!\langle M\rangle\!\rangle_{q_1}\}) \\
\equiv & (\nu q_1)([\![M]\!]_{q_1} \mid [\![P']\!]\{y := \langle\!\langle M\rangle\!\rangle_{q_1}\}) \\
\equiv & [\![P'\{y := M\}]\!]
\end{array}$$

The last two steps are derived, respectively, by the congruence $(\nu n)n[\mathbf{0}] \equiv \mathbf{0}$, and by the equivalence $[\![P\{x := M\}]\!] \equiv (\nu p)([\![M]\!]_p \mid [\![P]\!]\{x := \langle\!\langle M\rangle\!\rangle_p\})$, that can be shown to hold for the encoding.

6 Conclusions

We have investigated new mechanisms to protect migrating agents against the untrusted networks they traverse. Our primitives are best understood as low-level primitives to be employed for a secure implementation of the abstract mechanisms for secrecy found in mainstream ambient calculi such as the Ambient calculus.

The resulting calculus, SBA, is derived as a natural extension of NBA, the variant of Boxed Ambients studied in [7] which, in turn, borrowed the idea of capabilities with passwords from the work by Merro and Hennessy on Safe Ambients with Passwords (SAP) [16]. In fact, NBA can be encoded SBA simply by defining the capability $\mathsf{in}\langle n,k\rangle$ as $\mathsf{seal}\ k.\mathsf{in}\ n$, and similarly for $\mathsf{out}\langle n,k\rangle$. On the other hand, the sealing model of SBA appears provide strictly more flexibility, and expressive power, than the use of passwords in NBA: a SBA agent can be sealed by any of its local threads requesting a sealing Hence, an agent can be sealed and protected from undesired interactions by firing an action in one of local threads, and it not clear that a corresponding mechanism can be recovered in NBA (or SAP).

We have investigated the role of types in enforcing static guarantees of safety and secrecy in the presence of untyped opponents. It is worth remarking that even though our typing deals with untrusted networks, similar ideas can be used to generalise those presented in [6] and in [11] for access control and information flow security with untrusted components.

Similar studies have been conducted on other process calculi in the literature. In fact, our use of the trusted/untrusted rules is directly inspired by work on the pi/spi calculus (Cardelli [9], Gordon [13], Abadi [3]). Alternative approaches to the same problem have also been investigated. Among these, in [14] Hennessy and Riely study an extension of the Dπ-calculus with a type system that labels some location as untrusted and relies on run-time type checking to enforce security restrictions for processes coming from untrusted locations. Similar approaches have also been advocated for Mobile Ambients [4], and other calculi (notably Klaim).

Several questions remain to be explored, specifically, whether the selective data encryption underlying the sealing mechanisms we have introduce can be implemented effectively, and efficiently. Furthermore, in its current formulation, sealing an ambient has only the effect of guaranteeing the secrecy of data. More powerful mechanisms may be necessary to protect migrating agents by further hyding their structure or encrypting subcomponents consisting of data and code. Plans for future include work in both these directions.

Acknowledgements. We warmly thank Beppe Castagna for his insightful suggestions, and the referees of an earlier draft of the paper for their comments.

References

1. M. Abadi. Protection in programming-language translations. In *Automata, Languages and Programming: 25th International Colloquium, ICALP'98*, pages 868–883. Springer-Verlag, 1998.
2. M. Abadi and B. Blanchet. Analyzing security protocols with secrecy types and logic programs. In *Proceedings of POPL'02*, pages 33–44. ACM Press, 2002.
3. M. Abadi and A. Gordon. A Calculus for Cryptographic Protocols: The Spi Calculus. *Information and Computation*, 148(1):1–70, 1999.
4. M. Bugliesi and G. Castagna. Secure safe ambients. In *Proc. of the 28th ACM Symposium on Principles of Programming Languages*, pages 222–235, London, 2001. ACM Press.
5. M. Bugliesi, G. Castagna, and S. Crafa. Boxed ambients. In *TACS 2001 (4th. International Symposium on Theoretical Aspects of Comuter Science)*, number 2215 in Lecture Notes in Computer Science, pages 38–63, Sendai, Japan, 2001. Springer.
6. M. Bugliesi, G. Castagna, and S. Crafa. Reasoning about security in mobile ambients. In *CONCUR 2001 (12th. International Conference on Concurrency Theory)*, number 2154 in Lecture Notes in Computer Science, pages 102–120, Aalborg, Danemark, 2001. Springer.
7. M. Bugliesi, S. Crafa, M. Merro, and V. Sassone. Communication interference in mobile boxed ambients. In M. Agrawal and A. Seth, editors, *22nd Conference on Foundations of Software Technologies and Theoretical Computer Science, FST TCS 2002*, volume 2556 of *Lecture Notes in Computer Science*, pages 71–84. Springer, Kanpur, India, December 2002 2002.
8. L. Cardelli. Abstractions for mobile computations. In *Secure Internet Programming*, number 1603 in Lecture Notes in Computer Science, pages 51–94. Springer, 1999.
9. L. Cardelli, G. Ghelli, and A. D. Gordon. Secrecy and group creation. In *CONCUR2000:Concurrency Theory*, number 1877 in Lecture Notes in Computer Science, pages 365–379. Springer-Verlag, August 2000.
10. L. Cardelli and A. Gordon. Mobile ambients. In *FoSSaCS'98*, number 1378 in Lecture Notes in Computer Science, pages 140–155. Springer, 1998.
11. S. Crafa, M. Bugliesi, and G. Castagna. Information Flow Security for Boxed Ambients. *ENTCS*, 66(3), 2002.
12. C. Fournet, J-J. Levy, and Schmitt. A. An asynchronous, distributed implementation of mobile ambients. In *International Conference IFIP TCS*, number 1872 in Lecture Notes in Computer Science. Springer, 2000.
13. A. D. Gordon and A. Jeffrey. Authenticity by typing for security protocols with a. jeffrey. In *14th IEEE Computer Security Foundations Workshop (CSFW 2001)*, pages 145–159, Cape Breton, 2001. IEEE Computer Society.
14. M. Hennesy and J. Riely. Type–safe execution of mobile agents in anonymous networks. In J. Vitek and C. Jensen, editors, *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, number 1603 in LNCS. Springer, 1999.
15. F. Levi and D. Sangiorgi. Controlling interference in ambients. In *POPL2000*, pages 352–364. ACM Press, 2000.
16. M. Merro and M. Hennessy. Bisimulation Congruences in Safe Ambients. In *POPL'02 Proc. 29th ACM Symposium on Principles of Programming Languages*, pages 71–80. ACM Press, 2002.
17. R. Milner, J. Parrow, and D. Walker. A Calculus of Mobile Processes, Parts I and II. *Information and Computation*, 100:1–77, September 1992.
18. T. Sander and C. Tschudin. Towards mobile cryptography. In *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, USA, 1998. IEEE Computer Society Press.
19. D. Sangiorgi and A. Valente. A distributed abstract machine for safe ambients. In *Proc. of ICALP 2001*, pages 408–420, 2001.
20. C. Tschudin. Mobile agent security. In M. Klusch, editor, *Intelligent information agents: agent based information discovery and management in the Internet*, pages 431–446. Springer-Verlag, 1999.
21. U. G. Wilhelm, L. Buttyàn, and S. Staamann. On the problem of trust in mobile agent systems. In *Symposium on Network and Distributed System Security*. Internet Society, 1998.