Secrecy in Untrusted Networks*

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

[1] Università "Ca' Foscari", Venezia;
[2] Max-Planck-Institut für Informatik;
[3] University of Sussex

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 $(\boldsymbol{\nu} n)(\ \overline{n}\langle m\rangle \mid n(x).P\ )$ 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 them 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, as for instance the *encrypted* connection over a *public* channel of the spi calculus [3]:

$$(\boldsymbol{\nu} n)(\ \overline{p}\langle\{m\}_n\rangle \mid p(y).case\ y\ of\ \{x\}_n\ in\ P\ ) \tag{1}$$

The knowledge of n is still confined here, but its role is different: n is 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 of the shortcomings of the scheme.)

Similar mechanisms for secrecy are available for Mobile Ambients, MA, [10]. The following process, for instance, provides for the exchange of messages between locations a and b.

$$(\boldsymbol{\nu} n)(a[\ n[\,\mathsf{out}\ a.\mathsf{in}\ b.\langle m\rangle\,]\ ] \mid b[\,\mathsf{open}\ n.(x).P\,]) \tag{2}$$

* Research supported by EU FET-GC 'MyThS: Models and Types for Security in Mobile Distributed Systems' IST-2001-32617 and 'Mikado: Mobile Calculi based on Domains' IST-2001-32222, and by MIUR Project 'Modelli Formali per la Sicurezza'.

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 new flexible, lower-level mechanisms. Our work is inspired by the development of spi from π in the ambition of identifying suitable such primitives.

Structure of the paper. §1 discusses how to achieve secrecy in (variants of) MA, and presents the motivations for introducing specific primitives; §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 governs the interactions between trusted and untrusted (*opponents*) components travelling over open networks. Types split the world in two: the trusted *system* and the untrusted *context*. Relying on such intuition, data coming from the external environment is assigned an "unknown" type Public; Public values are handled with suspicion, since there is no saying what they are, or whom they are from. The type system guarantees a *secrecy* property proved in §4: a well-typed process does not disclose its secrets to any adversary, even though these may know its public names and traverse its sub-ambients. §5 presents an encoding of the spi calculus in SBA as a starting point for future comparisons between the two calculi.

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 mechanisms of authentication between such sites and encrypted agents on the move (cf. [20,18]). The sealing mechanism we envision aims at protecting the secrecy of data inside ambient-like mobile agents 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 centred 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, its content is protected from undesired access. The protocol for message exchange in (2), which here we question, is based on such secrecy assumption.

We start from the observation that ambient movement cannot occur without ambient revealing their names to some (not necessarily trusted) component of the underlying infrastructure. This happens in current implementations (cf. the ambient managers of [13] and the pointers-to-parent of [19]), and 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. Travelling 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 require such protocols to work as direct dialogues between the interested parties. This can only rely on direct communication and, therefore, force agents to reveal their interfaces to the network. Thus, quite as secure remote communication cannot rely exclusively on private names, the security of a mobile ambient cannot be relegated to the confidentiality of its name. Back to our example, the encapsulation mechanism of (2) turns out not to be secure, as in a realistic scenario name n will have to be disclosed.

We may think of two ways to provide for stronger security guarantees. One possibility is to commit to agents their own security, by resorting to co-capabilities. For instance, process (1) can be recast in Safe Ambients, SA, [16] as shown below.

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

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

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

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. Notice that we ignore here the orthogonal issue of authenticating b against possible malicious impersonators. In (4), name n and message m are protected by the wrapper ambient p, to be opened at the target site b. Now n need not reveal its name – even though p is forcibly opened by an attacker – because it does not move.

Whether or not these protocols are satisfactory depends on the kind of agents and networks targeted. If we look at ambients as abstract 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 co-capabilities in (3). If instead, we think of ambients as representing "soft" agents, then (3) is only appropriate in "friendly" networks, where gateways respect the privacy of the code they route to the next hop to destination.

The second approach is more robust and applies well to the case of soft agents. In particular, in (4), 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, which cannot move autonomously when encrypted.

The solution we advocate combines the benefits of the two approaches just discussed, by introducing new abstract primitives (which can be read as) providing for subjective access control by ways of co-capabilities, and 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 ambient owns 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 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 capability. This means that Q inside b has learnt the name of the incoming agent a. Observe that k acts to control access to b, and must be matched by a for the move to take place. Actually, name binding and access control checking work in a way at all analogous to the exchange of names and credentials which occur when registering for a networked service (cf. [7] for a deeper discussion and for related work).

Following the intuitions highlighted above, on top of the communication and movement 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, ambient n blocks all its interactions with the outside and encrypts all its messages (to be exchanged either locally or across boundaries), included those in the thread Q. The flexibility of this mechanism derives from the fact that a "sealed" ambient $n\{\![P \mid Q]\!\}_k$ is still (partially) active: in particular, it may still move over the network and perform limited forms of local synchronisation. On the contrary, its message exchanges are blocked and all its data encrypted, and so remain until it reaches a computational environment which knows k, the sealing key. The mechanism to unseal a sealed ambient is associated to movement and exercised through co-capabilities containing keys such as in the following process, where $n\{ P' \}$ is an ambient that can be either sealed or not:

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

The resulting model can, in some respects, be viewed as a symmetric cryptosystem, with encryption associated with the sealing capability that secures the 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 below is a proper extension of the syntax of Boxed Ambients, BA, [5], with movement co-capabilities and new 'sealing' primitives.

Expressions	M, N	$::=$	$k \cdots q \mid x \cdots z \mid \mathsf{in}\ M \mid \mathsf{out}\ M \mid \overline{\mathsf{in}} \mid \overline{\mathsf{out}} \mid M.M$
Locations	η	$::=$	$M \mid \uparrow \mid \star$
Prefixes	π	$::=$	$M \mid (x_1, \ldots, x_k)^{\eta} \mid \langle M_1, \ldots, M_k\rangle^{\eta} \mid \overline{\mathsf{in}}\ \{x\}_M \mid \overline{\mathsf{out}}\ \{x\}_M \mid \mathsf{seal}\ M$
Processes	P	$::=$	$\mathbf{0} \mid \pi.P \mid (\boldsymbol{\nu} n)P \mid P \mid P \mid\ !P \mid M[\![P]\!] \mid M\{\![P]\!\}_N$

Names $(k \cdots q)$ and variables $(x \cdots z)$ range over two disjoint sets; we use $a \cdots d$ to denote elements from either set, when the distinction is immaterial. Messages are formed as usual over names and (sequences of) capabilities. Locations indicate the target of a communication, i.e. a process in a child ambient M, in the parent ambient $(\uparrow)$, or a local process $(\star)$. 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 BA, and ambients, as in MA. 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 sealing $\mathsf{seal}\ k.P$.

We follow the usual conventions. Parallel composition has the lowest precedence among the operators, $\pi_1.\pi_2.P$ is read as $\pi_1.(\pi_2.P)$, while $\langle\tilde{M}\rangle^\eta, (\tilde{x})$ and $(\boldsymbol{\nu}\tilde{n})$ stand for $\langle M_1, \ldots, M_k\rangle^\eta, (x_1, \ldots, x_k)^\eta$, $(\boldsymbol{\nu}\ n_1, \ldots, n_k)$, 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 $\star$ for local communication, is omitted. The operators $(\boldsymbol{\nu}n)P$, $\overline{\mathsf{in}}\ \{x\}_a.P$, $\overline{\mathsf{out}}\ \{x\}_a.P$, 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; notice that in the following $M\{\ P\ \}$ always refers to the same kind of ambient on both the sides of a reduction rule.

Reduction. The operational semantics is defined as usual in terms of reduction and structural congruence. The definition of structural congruence is standard (cf. [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. An unsealed ambient is fully operational and may move, as well as communicate. The two states for reductions are formalised by defining the reduction relation in terms of two, inter-dependent relations, formalised in Table 1.

The relation $\dashrightarrow$ (referred to as *silent reduction*) gives the semantics of mobility and sealing. 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 cooperate by offering a co-capability. 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 it to the variable x. In addition, the authentication mechanism of SBA has the effect of removing the seal on the incoming ambient, so as to enable it to interact with the accepting context. Rule *(seal)* shows the effect of sealing: the capability $\mathsf{seal}\ k$ instructs a process to seal its enclosing ambient under a key k. Notice that encryption of individual messages remains indicated only implicitly by the $\{\![\ldots]\!\}$ around the ambient; besides being a convenient notation, this abstracts away from irrelevant implementation details.

Silent reductions may occur within any context, except under prefixes. On the contrary, the reductions involving communication – which are exactly as in previous versions of (N)BA, viz. [11,7] – may only occur within unsealed ambients, as formalised by the relation $\longrightarrow$. This reflects the fact that semantically relevant local communications must involve clear-text messages and, therefore, be avoided in untrusted environments, i.e. when ambients are sealed. Finally, rule (*struct*) is standard, while rules (*silent*) and (*ambient*) guarantee that the two reduction relations are linked properly.

Remarks. For ease of presentation, the syntax and operational semantics are so defined as to guarantee that ambients cannot be sealed more than once. An alternative choice would be to separate 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, together with rules (*enter*) and (*exit*), would implement an unsealing mechanism similar to ours, albeit not atomic. However, our proposal 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.

Examples. The kind of secure communication expected of the exchange of messages in (2) can now be achieved as in $(\boldsymbol{\nu} n)a[\![\ p[\![\,\mathsf{seal}\ n.\mathsf{out}\ a.\mathsf{in}\ b.\langle m\rangle^{\uparrow}\,]\!]\ ]\!] \mid \overline{\mathsf{out}} \mid b[\![\,\overline{\mathsf{in}}\ \{x\}_n.(y)^x.P \mid Q\,]\!]$. The public ambient p seals itself with the private key n, 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 the spi-calculus, it is the sealing key that is private, while the name of the ambient may be left public.

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, there is no guarantee that m will be received by the intended process when multiple threads are present inside b. 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, and easily be avoided 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 s_i of network sites and reports the names of those which provide a specific item it. To protect the agent moving over the network, we use the SBA primitives as follows. Let k be a sealing key shared between user u and sites s_i. The user can be represented as the process $\mathsf{u}[\![\,(\boldsymbol{\nu} a)a[\![\,P \mid R\,]\!] \mid Q\,]\!]$, where a is an agent with two threads: a router R, which controls movements, and a

Table 1. Reduction and Silent Reduction

Silent Reduction

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

Mobility (I)

(*enter*)	$n\{\mathsf{in}\ m.P \mid Q\} \mid m\{\overline{\mathsf{in}}\ .R \mid S\} \dashrightarrow m\{n\{P \mid Q\} \mid R \mid S\}$
(*exit*)	$m\{n\{\mathsf{out}\ m.P \mid Q\} \mid R\} \mid \overline{\mathsf{out}}\ .S \dashrightarrow n\{P \mid Q\} \mid m\{R\} \mid S$

Mobility (II)

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

Structural Rules

(*struct*)	$P \equiv Q, Q \dashrightarrow R,\ R \equiv S \Rightarrow P \dashrightarrow S$
(*context*)	$P \dashrightarrow Q \Rightarrow \mathbf{S}\{P\} \dashrightarrow \mathbf{S}\{Q\}$

Reduction

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

Communication

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

Structural Rules

(*silent*)	$P \dashrightarrow Q \Rightarrow P \longrightarrow Q$
(*ambient*)	$P \longrightarrow Q \Rightarrow n[P] \dashrightarrow n[Q]$
(*struct*)	$P \equiv Q, Q \longrightarrow R,\ R \equiv S \Rightarrow P \longrightarrow S$
(*context*)	$P \longrightarrow Q \Rightarrow \mathbf{E}\{P\} \longrightarrow \mathbf{E}\{Q\}$

communicator P, which interacts with the visited sites. We use two locks l and r to synchronise the two threads within a.

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

where $[a = b]P \triangleq (\boldsymbol{\nu} c)\ (c[\,\langle\ \rangle^a \mid b[\,(\ )^\uparrow.\langle\ \rangle^\uparrow\,] \mid (\ )^b.\langle\ \rangle^\uparrow\,] \mid (\ )^c.P), (c \notin \mathit{fn}(P))$
and $\mathsf{synch}(n) \triangleq n[\,n\{\!|\,\mathsf{out}\ n\,|\!\}_n\,], \overline{\mathsf{synch}}(n) \triangleq \overline{\mathsf{out}}\ \{\_\}_n$

The first thread is a loop that, when activated, seals the agents under the key k, 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, it waits for the sibling thread to seal the agent using k. The reduction semantics guarantees that, whenever the ambient a is not inside a site which knows k, all data in a are sealed, hence kept secret.

To synchronise with each other, the router and the communicator use the process forms: $\mathsf{synch}(n)$ and $\overline{\mathsf{synch}}(n)$. Interestingly, local synchronisation between threads is available even though the ambient is sealed, since it does not rely on exchanges of messages. Finally, each of the visiting sites can be coded as $\mathsf{s_i}[\, \overline{\mathsf{in}}\ \{z\}_k.(x)^z.\langle f_i(x), \mathsf{s_i}\rangle^z \mid \ldots]$. When 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 lookup performed by the site searching for item x: the result is x if $\mathsf{s_i}$ has x on sale, or some different value otherwise. Of course, rather than total unsealing, a policy of selective decryption of sensitive data may be desirable when agents interact with sites only partially trusted; this variation of the example can easily be implemented in SBA.

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 behaviour. Correspondingly, we assign a 'default' type Public to data that comes from untyped processes, and we handle such data carefully. The structure of types is defined by the following productions:

$$\begin{array}{lrl} \textit{Expression Types} & W ::= & \mathsf{Amb}[E] \mid \mathsf{Key}[E] \mid \mathsf{Public} \\ \textit{Exchange Types} & E, F ::= & \mathsf{shh} \mid (W_1, \ldots, W_k) \\ \textit{Process Types} & T ::= & [E, F] \mid \mathsf{Un} \end{array}$$

Untrusted processes are built upon expressions of type Public. In addition, the type Public is assigned to expressions that trusted processes may exchange with untrusted ones. 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 (trusted) ambients of type $\mathsf{Amb}[E]$. The latter, in turn, is the type of all the trusted ambients whose upward exchanges (if any) have type E. Notice that only ambients (not generic expressions) can be sealed. However, even untrusted ambients may be sealed, but in that case the sealing key is a generic expression of type Public and no security guarantee is made. 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 interaction between trusted and untrusted processes. Specifically, ambients have full migration capabilities as the type system allow trusted ambients to traverse untrusted ones and vice versa (as in the example of §2). Instead, a trusted (resp. untrusted) sealed ambient may be unsealed only within trusted (resp. untrusted) contexts. As for communication, the following policy is adopted: (i) local exchanges are allowed everywhere except that at top level, where we disallow local exchanges between trusted and untrusted processes, and (ii) trusted and untrusted ambients may exchange values across boundaries, provided that such values have type Public. We proceed with the description of the typing rules, collected in Tables 2 and 3.

Typing Rules. Every (co-)capability is assigned type Public; accordingly, 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, we will prove that the type system providse the expected guarantees of secrecy and safety for any value exchange.

Each process form has two associated typing rules, depending on whether the process in question is to be considered trusted (Table 2) or deemed untrusted (Table 3): 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 need four rules: rule (Amb Seal) in Table 2, assigns a type to ambients formed with the 'right' key N, and enclosing a process P with the expected exchanges. Rule (Amb) in Table 2 is standard. Rules (Untrusted Amb/Amb Seal), in Table 3 are used to type untrusted, possibly ill-formed, ambients. In addition, observe that a trusted (sealed) ambient 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. As we noted above, we allow untrusted and trusted process to exchange values, as long as these have type Public, as required in rules (Untrusted Input/Output M) in Table 3. Note also that in these rules we do not require that the arity of the downward communication matches that of the target ambient. This leaves full flexibility in the typing of opponent processes, as it is implied by the following proposition.

Proposition 1 (Typability). *Let P be a process with $fn(P) = \{a_1, \ldots, a_n\}$ and $fv(P) = \{x_1, \ldots, x_m\}$. Then $a_1 : \mathsf{Public}, \ldots, a_n : \mathsf{Public}, x_1 : \mathsf{Public}, \ldots, x_m : \mathsf{Public} \vdash P : \mathsf{Un}$,*

In other words, no constraint is imposed on the structure of the opponent: only that it initially does not know any secret. In addition, one can easily prove the standard property of type preservation under reduction.

Proposition 2 (Subject Reduction). *If $\Gamma \vdash P : T$ and $P \longrightarrow Q$, then $\Gamma \vdash Q : T$.*

Table 2. Typing Rules: Trusted Processes

$$\text{(EMPTY)}\ \frac{}{\emptyset \vdash \diamond} \qquad \text{(ENV } x\text{)}\ \frac{\Gamma \vdash \diamond \quad x \notin \mathit{Dom}(\Gamma)}{\Gamma, x : W \vdash \diamond} \qquad \text{(PROJECTION)}\ \frac{\Gamma \vdash \diamond \quad \Gamma(M) = W}{\Gamma \vdash M : W} \qquad \text{(PATH)}\ \frac{\Gamma \vdash M_1 : \mathsf{Public} \quad \Gamma \vdash M_2 : \mathsf{Public}}{\Gamma \vdash M_1.M_2 : \mathsf{Public}}$$

$$\text{(CO-IN)}\ \frac{\Gamma \vdash \diamond}{\Gamma \vdash \overline{\mathsf{in}} : \mathsf{Public}} \qquad \text{(CO-OUT)}\ \frac{\Gamma \vdash \diamond}{\Gamma \vdash \overline{\mathsf{out}} : \mathsf{Public}} \qquad \text{(IN } M\text{)}\ \frac{\Gamma \vdash M : W \quad W \in \{\mathsf{Amb}[E], \mathsf{Public}\}}{\Gamma \vdash \mathsf{in}\ M : \mathsf{Public}}$$

$$\text{(OUT } M\text{)}\ \frac{\Gamma \vdash M : W \quad W \in \{\mathsf{Amb}[E], \mathsf{Public}\}}{\Gamma \vdash \mathsf{out}\ M : \mathsf{Public}} \qquad \text{(PREFIX)}\ \frac{\Gamma \vdash M : \mathsf{Public} \quad \Gamma \vdash P : T}{\Gamma \vdash M.P : T} \qquad \text{(SEAL)}\ \frac{\Gamma \vdash M : \mathsf{Key}[E] \quad \Gamma \vdash P : [F, E]}{\Gamma \vdash \mathsf{seal}\ M.P : [F, E]}$$

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

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

$$\text{(DEAD)}\ \frac{\Gamma \vdash \diamond}{\Gamma \vdash \mathbf{0} : T} \qquad \text{(PAR)}\ \frac{\Gamma \vdash P : T \quad \Gamma \vdash Q : T}{\Gamma \vdash P \mid Q : T} \qquad \text{(NEW)}\ \frac{\Gamma, n : W \vdash P : T}{\Gamma \vdash (\boldsymbol{\nu} n)P : T} \qquad \text{(REPL)}\ \frac{\Gamma \vdash P : T}{\Gamma \vdash !P : T}$$

$$\text{(LOCAL INPUT)}\ \frac{\Gamma, x_1 : W_1, \ldots, x_k : W_k \vdash P : [(W_1, \ldots, W_k), E]}{\Gamma \vdash (x_1, \ldots, x_k).P : [(W_1, \ldots, W_k), E]} \qquad \text{(LOCAL OUTPUT)}\ i = 1, \ldots, k\ \frac{\Gamma \vdash M_i : W_i \quad \Gamma \vdash P : [\tilde{W}, E]}{\Gamma \vdash \langle \tilde{M} \rangle.P : [\tilde{W}, E]}$$

$$\text{(INPUT } \uparrow\text{)}\ \frac{\Gamma, x_1 : W_1, \ldots, x_k : W_k \vdash P : [E, (W_1, \ldots, W_k)]}{\Gamma \vdash (x_1, \ldots , x_k)^{\uparrow}.P : [E, (W_1, \ldots, W_k)]} \qquad \text{(OUTPUT } \uparrow\text{)}\ i = 1, \ldots, k\ \frac{\Gamma \vdash M_i : W_i \quad \Gamma \vdash P : [E, (W_1, \ldots, W_k)]}{\Gamma \vdash \langle M_1, \ldots, M_k \rangle^{\uparrow}.P : [E, (W_1, \ldots, W_k)]}$$

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

4 A Secrecy Theorem

We refer to a 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]. We adapt that definition to our framework by representing an attacker as a closed, but otherwise arbitrary, context. 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 characterise the initial knowledge of the attacker in terms of the names, the keys and the capabilities initially known to it. Interestingly, the knowledge of

Table 3. 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 Cap)
$$\frac{\Gamma \vdash M : \mathsf{Public} \quad \Gamma \vdash P : \mathsf{Un}}{\Gamma \vdash M.P : \mathsf{Un}}$$

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

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

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

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

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

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

capabilities is important here, since by exercising (a sequence 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 define a context $\mathbf{A}(-)$ to be a process that contains exactly one variable $(-)$ (i.e. a hole). We denote with $\mathbf{A}(P)$ the process resulting from substituting the variable with P in A. Also, 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 free in P, and S a finite set of names and capabilities.* P may reveal n to S *iff there exists an S-adversary $\mathbf{A}(-)$, with $\mathbf{A}(P)$ 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 n to S.*

The definition extends readily to private names as follows (cf. [9]): $(\boldsymbol{\nu} n)P$ *may reveal* n *to* S if and only if there is a fresh name m such that $P\{n := m\}$ may reveal m to S, with $m \notin S \cup fn(P)$. Notice that an adversary may dynamically acquire new names and new capabilities (i) by creating its own fresh names, (ii) by receiving names over public channels, and (iii) 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 capability to move in and out any ambient of P (included its secret ambients).

Theorem 1 (Secrecy). *Let P be a process such that $\Gamma \vdash P : \mathsf{Un}$ 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.*

Notice that the theorem only holds for well-typed processes of type Un. This immediately rules out processes that exchange non-public data at top level. Indeed, for such processes no secrecy guarantee can be made, for adversaries always have free access to the anonymous top level channel of any process. On the other hand, the theorem captures precisely the security guarantees our approach was intended to provide. That follows by observing (i) that well-typed ambient processes can always be typed with type Un, and (ii) that ambients (i.e. agents) are indeed the objects of our security concerns.

5 Encoding of the Spi Calculus

We further illustrate the calculus with an encoding of spi-calculus [3]. To ease the presentation, we focus on the following fragment of the asynchronous spi-calculus, in which we disregard the construct for pairs, natural numbers and matching.

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

Table 4. Encoding of the spi calculus

$$\langle\!\langle\, a \,\rangle\!\rangle_p = a \qquad \text{(for } a \text{ a name or a variable)}$$

$$\langle\!\langle\, \{M_1, \dots, M_n\}_k \,\rangle\!\rangle_p = p$$

$$\llbracket\, a \,\rrbracket_p = \mathbf{0} \qquad \text{(for } a \text{ a name or a variable)}$$

$$\begin{aligned}\llbracket\, \{M_1, \dots, M_n\}_k \,\rrbracket_p = (\boldsymbol{\nu} q_1, \dots, q_n) \qquad & \{q_1, \dots, q_n\} \cap (fn(M_1, \dots, M_n) \cup \{k, p\}) = \emptyset \\ \llbracket\, M_1 \,\rrbracket_{q_1} \mid \dots \mid \llbracket\, M_n \,\rrbracket_{q_n} \mid\, !\, p[\, (x)^{\uparrow}.\mathsf{seal}\ k.\mathsf{in}\ x.\langle \langle\!\langle\, M_1 \,\rangle\!\rangle_{q_1}, \dots, \langle\!\langle\, M_n \,\rangle\!\rangle_{q_n} \rangle^{\uparrow}\,]\,) &\end{aligned}$$

$$\llbracket \mathbf{0} \rrbracket = \mathbf{0}, \qquad \llbracket (\boldsymbol{\nu} n)P \rrbracket = (\boldsymbol{\nu} n)\llbracket P \rrbracket, \qquad \llbracket P \mid Q \rrbracket = \llbracket P \rrbracket \mid \llbracket Q \rrbracket$$

$$\begin{aligned}\llbracket\, \bar{b}\langle M_1, \dots, M_n \rangle \,\rrbracket = (\boldsymbol{\nu} q_1, \dots, q_n) \qquad & \{q_1, \dots, q_n\} \cap (fn(M_1, \dots, M_n) \cup \{b\}) = \emptyset \\ \llbracket\, M_1 \,\rrbracket_{q_1} \mid \dots \mid \llbracket\, M_n \,\rrbracket_{q_n} \mid b[\, \langle \langle\!\langle\, M_1 \,\rangle\!\rangle_{q_1}, \dots, \langle\!\langle\, M_n \,\rangle\!\rangle_{q_n} \rangle^{\uparrow}\,] &\end{aligned}$$

$$\llbracket\, b(x_1, \dots, x_n)P \,\rrbracket = (x_1, \dots, x_n)^b.\llbracket\, P \,\rrbracket$$

$$\begin{aligned}\llbracket\, \mathit{case}\ M\ \mathit{of}\ \{x_1, \dots, x_n\}_k\ \mathit{in}\ P \,\rrbracket = (\boldsymbol{\nu} p)(\llbracket\, M \,\rrbracket_p \mid (\boldsymbol{\nu} c)((\langle c \rangle^{\langle\!\langle\, M \,\rangle\!\rangle_p} \mid \qquad & \{p, c\} \cap \{fn(P) \cup fn(M)\} = \emptyset \\ c[\, \overline{\mathsf{in}}\ \{y\}_k.(x_1, \dots, x_n)^y.\langle x_1, \dots, x_n \rangle^{\uparrow}\,] \mid (x_1, \dots, x_n)^c.\llbracket\, P \,\rrbracket)) &\end{aligned}$$

The operational semantics of this fragment is standard (cf. [3]): in particular, decryption 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 message is then accounted for by communicating the name of the corresponding ambient. The formal definition is given in Table 4 in terms of three translation maps: $\langle\!\langle \cdot \rangle\!\rangle_p$: *Expressions* $\mapsto$ *Expressions*, $\llbracket \cdot \rrbracket_p$: *Expressions* $\mapsto$ *Processes*, and $\llbracket \cdot \rrbracket$: *Processes* $\mapsto$ *Processes*. In the first two (subsidiary) maps, p is the name of the ambient (if any) enclosing the message to be exchanged. In particular, if M is a name or a variable, then $\langle\!\langle\, M \,\rangle\!\rangle_p$ returns M; if instead M is an encryption packet, $\llbracket\, M \,\rrbracket_p$ returns p, the name of the ambient that stores the packet. Correspondingly, $\llbracket\, M \,\rrbracket_p$ 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 $\llbracket\, M \,\rrbracket_p$ first reads a name x, then gets sealed with k to move into x, where eventually gets unsealed and delivers its payload. The use of replication on the ambient encoding an encryption packet accounts for the possible non-linear usage of messages in spi.

The encoding can be shown to be sound with respect to appropriate choices of behavioural equivalences in the two calculi, noted $\cong_{spi}$ and $\cong_{SBA}$, respectively. In particular, we take $\cong_{spi}$ to be *testing* equivalence, the notion of equivalence for spi-calculus studied in [3], and for SBA, we define $\cong_{SBA}$ to be reduction barbed congruence, based on the following exhibition predicate: $P \downarrow_b^{SBA} \triangleq P \equiv (\boldsymbol{\nu}\tilde{n})((\tilde{x})^b P_1 \mid P_2)$. Given these choices, one can prove that the encoding is equationally sound.

Theorem 2 (Soundness of the encoding). *If* $\llbracket\, P \,\rrbracket \cong_{SBA} \llbracket\, Q \,\rrbracket$ *then* $P \cong_{spi} Q$.

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.

The resulting calculus, SBA, is derived as a natural extension of NBA, the variant of Boxed Ambients studied in [7]. In fact, NBA can be interpreted into SBA by defining the capability $\mathsf{in}\langle n, k\rangle$ as $\mathsf{seal}\ k.\mathsf{in}\ n$, and similarly for $\mathsf{out}\langle n, k\rangle$. (Observe though that this lacks the atomicity of movement and credential verification of NBA.) On the other hand, the sealing model of SBA appears to provide strictly more flexibility and expressiveness than the access control of NBA: a SBA agent can be sealed by any of its local threads. Hence, an agent can be sealed and protected from undesired interactions by firing an action in one of its local threads, and it is not clear that a corresponding mechanism can be recovered in NBA.

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 π/spi calculus (Cardelli et al. [9], Gordon and Jeffrey [14], Abadi and Gordon [3]). Alternative approaches to the same problem have also been investigated. Among these, Hennessy and Riely [15] 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 [12]).

Several questions remain to be explored, as for instance whether the data encryption underlying the sealing mechanisms we have introduced 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 hiding their structure or encrypting subcomponents consisting of data and code. Plans for future include work in both these directions.

Acknowledgements. We would like to thank Beppe Castagna for his suggestions, and the anonymous referees for their comments.

References

1. M. Abadi. Protection in programming-language translations. In *Proceedings of ICALP'98*, number 1443 in LNCS, 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 *Procedings of POPL'01*, pages 222–235. ACM Press, 2001.
5. M. Bugliesi, G. Castagna, and S. Crafa. Boxed ambients. In *Proceedings of TACS'01*, number 2215 in LNCS, pages 38–63. Springer-Verlag, 2001.
6. M. Bugliesi, G. Castagna, and S. Crafa. Reasoning about security in mobile ambients. In *Proceedings of CONCUR 2001*, number 2154 in LNCS, pages 102–120. Springer-Verlag, 2001.
7. M. Bugliesi, S. Crafa, M. Merro, and V. Sassone. Communication interference in mobile boxed ambients. In *FST&TCS 2002*, volume 2556 of *LNCS*, pages 71–84. Springer-Verlag, 2002.
8. L. Cardelli. Abstractions for mobile computations. In *Secure Internet Programming*, number 1603 in LNCS, pages 51–94. Springer-Verlag, 1999.
9. L. Cardelli, G. Ghelli, and A. D. Gordon. Secrecy and group creation. In *Proceedings of CONCUR'00*, number 1877 in LNCS, pages 365–379. Springer-Verlag, August 2000.
10. L. Cardelli and A. Gordon. Mobile ambients. In *FoSSaCS'98*, number 1378 in LNCS, pages 140–155. Springer-Verlag, 1998.
11. S. Crafa, M. Bugliesi, and G. Castagna. Information Flow Security for Boxed Ambients. *ENTCS*, 66(3), 2002.
12. R. De Nicola, G. Ferrari, and R. Pugliese. Klaim: a kernel language for agents interaction and mobility. *IEEE Transactions on Software Engeneering*, 24:315–330, 1998.
13. C. Fournet, J-J. Levy, and Schmitt. A. An asynchronous, distributed implementation of mobile ambients. In *Proceedings of IFIP TCS'00*, number 1872 in LNCS. Springer-Verlag, 2000.
14. A. D. Gordon and A. Jeffrey. Authenticity by typing for security protocols. In *Proceedings of CSFW 2001*, pages 145–159. IEEE Computer Society, 2001.
15. M. Hennesy and J. Riely. Type–safe execution of mobile agents in anonymous networks. In *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, number 1603 in LNCS, pages 95–115. Springer-Verlag, 1999.
16. F. Levi and D. Sangiorgi. Controlling interference in ambients. In *Proceedings of POPL'00*, pages 352–364. ACM Press, 2000.
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.* IEEE Computer Society Press, 1998.
19. D. Sangiorgi and A. Valente. A distributed abstract machine for safe ambients. In *Proc. of ICALP 2001*, pages 408–420, 2001.
20. 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.