A Calculus for Secure Mobility

Bruno Blanchet[1] and Benjamin Aziz[2]

[1] CNRS, Département d'Informatique
École Normale Supérieure, Paris
and Max-Planck-Institut für Informatik
Bruno.Blanchet@ens.fr

[2] School of Computer Applications
Dublin City University
baziz@computing.dcu.ie

Abstract. In this paper, we introduce the crypto-loc calculus, a calculus for modelling secure mobile computations that combine the concepts of locations, cryptography, and code mobility. All these concepts exist in mobile systems, for example, Java applets run within sandboxes or downloaded under an SSL connection. We use observational equivalence of processes as a powerful means of defining security properties, and characterize observational equivalence in terms of a labelled bisimilarity relation, which makes its proof much easier.

1 Introduction

The issue of security has become central in mobile computing systems as a result of the increasing technological advances enjoyed in recent years and the appearance of new security threats associated with such advances. In process algebraic models of mobility, the modelling of the various actions that the system can perform has been captured in terms of a few important concepts. Among these are the concepts of *location*, *cryptography*, and *code mobility*. In most occasions, these concepts have been studied separately. Here, we combine the three concepts into one model with the goal of capturing the security and mobility features that already exist in platforms like Java and .NET, in a simple and elegant calculus. This combination makes it possible to analyze the security properties of cryptographic protocols also involving mobility, for instance.

Locations have significant security effects; they are name-protected areas of computation, which provide a handy isolation mechanism often useful in the modelling of concepts like firewalls and sandboxes. Communications between locations are generally restricted to either communications local to one location (ambients [12]), or communications with locations immediately inside or outside a location (boxed ambients [10]). The former implies using an opening mechanism to destroy ambient boundaries, in order to allow communication between ambients. This weakens the isolation power of locations considerably and has security drawbacks already recognized in previous work [10], so we prefer the latter choice.

The mobility of locations may be classified into *subjective* and *objective*. Subjective mobility (as in, e.g., the ambients calculus [12]) implies the presence of language primitives that can be used by a location as commands to directly control its movement. On the other hand, objective mobility (as in, e.g., the seal calculus [20]) dictates that locations be controlled by their context, which has primitives expressive enough to be able to send and receive locations. Furthermore, one can distinguish *weak* mobility, in which the received code always restarts its execution in the initial state, *strong* mobility, where a thread is sent along with its execution state, and *full* mobility, where a full program is sent along with its state [8]. In our model, certain kinds of data represent code. These data can be manipulated like any other piece of data: they can be sent and received (which provides *weak objective* mobility of any code, not just locations), but also manipulated by cryptographic primitives. In contrast to ordinary data, they can also be executed. This modelling of mobility corresponds to what happens in systems such as Java and .NET, and is also easier to implement than strong or full mobility.

Our representation of cryptography is borrowed from the applied pi calculus [3], in which cryptographic primitives are defined as functions satisfying an equational theory. This provides a generic treatment of primitives, which promotes a uniform understanding of the underlying theory and yields a powerful technique for specifying cryptographic systems.

As often in process calculi, some of these features can be encoded. Location-based languages can encode cryptographic operations, including symmetric and asymmetric encryption [12, 20]. (That would probably be more difficult for primitives such as xor.) Strong mobility can be encoded from weak mobility [8]. The execution of certain kinds of data can be encoded by an interpreter (such an interpreter has already been defined for the pi calculus [14]), and strong mobility could be encoded by an improved version of such an interpreter. Even if encodings exist, we choose to provide as primitives in our calculus the features that exist in Java and .NET, i.e. cryptography, sandboxes modelled by locations, and weak code mobility. It is obviously possible to encode other features in our calculus, such as strong mobility, much like one could encode them in Java.

The combination of locations, cryptography, and weak code mobility enables us to model scenarios, such as Java applets, in which code is sent on the network after a cryptographic treatment (signed applets) or inside a cryptographic protocol (applets sent over SSL, for instance). Based on the amount of trust established for an applet, it is then possible to run it inside or outside a sandbox (location). Similarly, we could also model applets that execute a cryptographic protocol, in which case the transmitted code uses cryptographic primitives. We develop a general technique based on observational equivalence to prove security properties, and use it to show some of the main properties of such examples.

Outline. In Sect. 2, we present the syntax and operational semantics of the calculus. In Sect. 3, we introduce and relate observational equivalence and labelled bisimilarity. In Sect. 4, we introduce a few examples of systems that could benefit from the calculus and discuss their security properties. In Sect. 5, we compare with related work and finally, in Sect. 6, we conclude.

2 Process Calculus

The syntax of the crypto-loc calculus is defined in Fig. 1. In this calculus, terms M, N represent passive data such as messages. These are either atomic terms, like *names* a, b, c and *variables* x, y, z, or complex terms that result from the application of functions f defined by a signature Σ. Names represent atomic data, such as keys, while variables will later be substituted by some possibly complex message. The signature Σ consists of a finite set of function symbols, with an associated arity. It is equipped with an equational theory, that is, a congruence relation on terms closed under substitutions of terms for free variables and names. We denote by $\Sigma \vdash M = N$ the equality of M and N modulo the equational theory, and we assume that there exist two different terms modulo this equational theory.

The signature Σ that we adopt in this paper mainly concentrates on the cryptographic functions, such as symmetric and asymmetric encryption/decryption, digital signatures/verification etc. For instance, shared-key cryptography can be defined by two functions **encrypt** for encryption and **decrypt** for decryption, with the equation $\mathbf{decrypt}(\mathbf{encrypt}(x, y), y) = x$. Data structures can also be encoded by including functions for tuple creation and projections. Sometimes we abbreviate tuples $(M_1, \ldots, M_n)$ as the vector $\vec{M}$. While, in this paper, our examples use only simple cryptographic primitives, the technique can also model complex cases such as xor and Diffie-Hellman key agreements [3]. The term $\mathbf{pack}(\vec{M}, \lambda\vec{x}.P)$ transforms a process P and a tuple $\vec{M}$ into data that can be treated as any other passive term. This removes the need for special input/output actions for processes. We postpone the detailed explanation of this term to after the explanation of processes.

The syntax of processes P, Q is described as follows. A null process 0 is an inactive process unable to evolve any further. The process $P \mid Q$ denotes the parallel composition of two process P and Q. A replicated process $!P$ spawns an unbounded number of copies of P. Restriction $(\nu a)P$ localizes the name a within the scope of P. A location $M[P]$ embodies the process P. The process P is said to *reside* at M. As in boxed ambients [10], an input action $M^{\eta}(\lambda x).P$ has three types: local-input $M^{\star}(\lambda x).P$, up-input $M^{\uparrow}(\lambda x).P$ and down-input $M^{N}(\lambda x).P$. A local input can only synchronize with output actions that are performed by processes local to the current location. Up-input can only receive messages from processes resident in the environment location enclosing the current location. A down-input $M^{N}(\lambda x).P$ can only receive messages from the sublocation N enclosed by the current location. Similarly, output actions $\overline{M}^{\eta}(M').P$ are divided into local-, up- and down-outputs. For instance, in the process $c[a^{\uparrow}(\lambda y).P] \mid \overline{a}^{c}(M).Q$, the down-output $\overline{a}^{c}(M)$ can send its message to the up-input $a^{\uparrow}(\lambda y)$ inside location c. The terms that represent channels in input/output actions, their η parameter, and the names of locations must reduce to names at runtime.

$M, N ::=$	terms
$\quad x, y, z$	$\quad$ variable
$\quad a, b, c, k$	$\quad$ name
$\quad f(M_1, \ldots, M_n)$	$\quad$ function application
$\quad \mathbf{pack}(\vec{M}, \lambda\vec{x}.P)$	$\quad$ processes (with $fn(P) = \emptyset, fv(P) \subseteq \{\vec{x}\}$)
$\eta ::=$	communication target
$\quad M$	$\quad$ down
$\quad \uparrow$	$\quad$ up
$\quad \star$	$\quad$ local
$P, Q ::=$	processes
$\quad 0$	$\quad$ null
$\quad P \mid Q$	$\quad$ parallel composition
$\quad !P$	$\quad$ replication
$\quad (\nu a)P$	$\quad$ restriction
$\quad M^{\eta}(\lambda x).P$	$\quad$ input
$\quad \overline{M}^{\eta}(M').P$	$\quad$ output
$\quad M[P]$	$\quad$ location
$\quad \mathbf{exec}(M)$	$\quad$ execute packed process
$\quad$ *if* $M = N$ *then* P *else* Q	$\quad$ conditional

Fig. 1. Syntax of the process calculus

Otherwise, the process blocks. The $\mathbf{exec}(M)$ process executes M when M is a **pack**ed process. Otherwise, it blocks. Finally, a conditional process compares two terms, M and N, and branches to either P or Q, depending on whether $\Sigma \vdash M = N$ holds true or not. As usual, we may omit an *else* clause when it consists of 0.

The substitution that replaces M for x is denoted by $\{M/x\}$. We extend this to tuples $\{\vec{M}/\vec{x}\}$. We denote by $P\sigma$ (resp. $M\sigma$) the application of the substitution σ to process P (resp. term M). We define *let* $x = M$ *in* P as syntactic sugar for $P\{M/x\}$. We denote by $\{\vec{x}\}$ the set of variables in the tuple $\vec{x}$. We denote by *fn* the free names and by *fv* the free variables of a process or a term, defined as usual. In $\mathbf{pack}(\vec{M}, \lambda\vec{x}.P)$, the variables $\vec{x}$ are bound. A process or a term is *closed* when it contains no free variable. It may have free names.

Let us now explain packed processes. Intuitively, in $\mathbf{pack}(\vec{M}, \lambda\vec{x}.P)$, P represents the code of the process and the tuple $\vec{M}$ represents its data. When executing such a process, the tuple $\vec{M}$ is substituted for the variables $\vec{x}$ in P, and the resulting process is executed. We include a function $\mathbf{getdata} \in \Sigma$ used (by malicious processes) to extract the data $\vec{M}$, defined by $\mathbf{getdata}(\mathbf{pack}(\vec{M}, \lambda\vec{x}.P)) = \vec{M}$.

The separation between data and code is important for security considerations, since an adversary can extract all data carried by a process. So, for example, the process:

$$\mathbf{pack}((s, k, a), \lambda(x, y, z).\overline{z}^{\uparrow}(\mathbf{encrypt}(x, y)))$$

reveals s, k, a when it is sent in clear over the network, whereas the process:

$$\mathbf{pack}((\mathbf{encrypt}(s,k),a),\lambda(x,z).\overline{z}^{\uparrow}(x))$$

reveals only $\mathbf{encrypt}(s,k)$ and a, but not s when the adversary does not have k. Then, these two processes must be represented differently, even if they perform the same actions when they are executed. The separation of code and data is enforced by the conditions $fn(P)=\emptyset$ and $fv(P)\subseteq\{\vec{x}\}$ that prevent the presence of data in P. The equational theory is such that $\Sigma\vdash\mathbf{pack}(\vec{M},\lambda\vec{x}.P)=\mathbf{pack}(\vec{M}',\lambda\vec{x}'.P')$ if and only if $\Sigma\vdash\vec{M}=\vec{M}'$ and $P'\{\vec{x}/\vec{x}'\}\equiv P$ (with $fn(P)=fn(P')=\emptyset$, $fv(P)\subseteq\{\vec{x}\}$, and $fv(P')\subseteq\{\vec{x}'\}$), where $\equiv$ is the structural equivalence defined below.

The adversary can also observe the code of packed processes, by testing equality with processes it builds. Hence, the process:

$$c^{\eta}(\lambda x).\mathit{if}\ x=\mathbf{pack}(\mathbf{getdata}(x),\lambda\vec{y}.P)\ \mathit{then}\ \ldots\ \mathit{else}\ \ldots$$

tests whether the code of x (received on c) is structurally equivalent to P (since the equality is defined as structural equivalence for processes). We could optionally add other functions to manipulate and transform the code of packed processes, but **getdata** already gives enough power to the adversary. Indeed, the adversary can build any process, so it can rebuild the code of packed processes without having a primitive to extract it. (Note, however, that scoping constraints introduced by the restriction prevent the adversary from using private data of processes, when it does not learn them by listening to communications.)

The formal semantics of the crypto-loc calculus is given in Fig. 2. Note that all rules apply only to closed processes and that in (11), M and M' must be closed. Processes are considered equal modulo renaming of bound names and variables. Processes are prepared syntactically for communications by applying the structural equivalence relation $\equiv$, which is defined as the least equivalence satisfying the first set of rules of Fig. 2. The second set of the rules of Fig. 2 introduces the reduction relation $\rightarrow$. The first three rules (Red Local), (Red Down), and (Red Up) deal with local, downward, and upward communications with respect to the process performing the output action. Rule (Red Exec) deals with the execution of packed processes. Other rules are standard.

As an example, let us consider the following process:

$$\begin{aligned}&(\nu k)(\overline{c}^{\star}(\mathbf{pack}((d,\mathbf{encrypt}(m,k)),\lambda(x_d,x_e).x_d{}^{\uparrow}(\lambda z).\overline{x_d}^{\star}(\mathbf{decrypt}(x_e,z))))\\&\quad\mid c^{\star}(\lambda y).b[\mathbf{exec}(y)\mid d^{\star}(\lambda x_m).0]\mid\overline{d}^{b}(k))\end{aligned}$$

This process first creates a new key k, shared between the processes that execute in parallel. The first element of the parallel composition outputs a packed process on the public channel c; the second one receives it, and executes it inside a location b. After these steps, the process becomes

$$(\nu k)(0\mid b[d^{\uparrow}(\lambda z).\overline{d}^{\star}(\mathbf{decrypt}(\mathbf{encrypt}(m,k),z))\mid d^{\star}(\lambda x_m).0]\mid\overline{d}^{b}(k))$$

Structural equivalence $P \equiv P'$:

$$P \mid 0 \equiv P \quad (1)$$
$$P \mid Q \equiv Q \mid P \quad (2)$$
$$(P \mid Q) \mid R \equiv P \mid (Q \mid R) \quad (3)$$
$$P \equiv Q \;\Rightarrow\; P \mid R \equiv Q \mid R \quad (4)$$
$$P \equiv Q \;\Rightarrow\; (\nu a)P \equiv (\nu a)Q \quad (5)$$
$$P \equiv Q \;\Rightarrow\; M[P] \equiv M[Q] \quad (6)$$
$$!P \equiv P \mid !P \quad (7)$$
$$(\nu a_1)(\nu a_2)P \equiv (\nu a_2)(\nu a_1)P \quad (8)$$
$$(\nu a)(P \mid Q) \equiv P \mid (\nu a)Q \text{ if } a \notin fn(P) \quad (9)$$
$$(\nu a)M[P] \equiv M[(\nu a)P] \text{ if } a \notin fn(M) \quad (10)$$
$$P\{M/x\} \equiv P\{M'/x\} \text{ if } \Sigma \vdash M = M' \quad (11)$$

Reduction $P \rightarrow P'$:

$$a^\star(\lambda y).P' \mid \overline{a}^\star(M).Q' \rightarrow P'\{M/y\} \mid Q' \quad \text{(Red Local)}$$
$$c[a^\uparrow(\lambda y).P' \mid R] \mid \overline{a}^c(M).Q' \rightarrow c[P'\{M/y\} \mid R] \mid Q' \quad \text{(Red Down)}$$
$$a^c(\lambda y).P' \mid c[\overline{a}^\uparrow(M).Q' \mid R] \rightarrow P'\{M/y\} \mid c[Q' \mid R] \quad \text{(Red Up)}$$
$$\mathbf{exec}(\mathbf{pack}(\vec{M}, \lambda\vec{x}.P)) \;\rightarrow\; P\{\vec{M}/\vec{x}\} \quad \text{(Red Exec)}$$
$$\textit{if } M = M \textit{ then } P \textit{ else } Q \;\rightarrow\; P \quad \text{(Red Then)}$$
$$\textit{if } M = N \textit{ then } P \textit{ else } Q \;\rightarrow\; Q \text{ if } \Sigma \not\vdash M = N \quad \text{(Red Else)}$$
$$P \;\rightarrow\; Q \;\Rightarrow\; P \mid R \;\rightarrow\; Q \mid R \quad \text{(Red Par)}$$
$$P \;\rightarrow\; Q \;\Rightarrow\; (\nu a)P \;\rightarrow\; (\nu a)Q \quad \text{(Red Res)}$$
$$P \;\rightarrow\; Q \;\Rightarrow\; M[P] \;\rightarrow\; M[Q] \quad \text{(Red Loc)}$$
$$P' \equiv P,\, P \;\rightarrow\; Q,\, Q \equiv Q' \;\Rightarrow\; P' \;\rightarrow\; Q' \quad \text{(Red} \equiv\text{)}$$

Fig. 2. Structural equivalence and reduction relations

The process that comes from the packed process waits for a message on channel d from outside location b. So it can receive the key k sent by the last element of the parallel composition. After receiving this key, it decrypts the message $\mathbf{encrypt}(m, k)$ with it, and outputs the result m on channel d locally inside location b. The message m is then received by $d^\star(\lambda x_m).0$.

3 Process Equivalences

Observational equivalence is a powerful means for defining security properties of processes and for reasoning about them. In this section, we define observational equivalence and show that a labelled bisimilarity relation can be used to provide simpler proofs of observational equivalence. Thus, we extend the results proved in [3] for the applied pi calculus to locations and mobility.

Intuitively, two processes are observationally equivalent when an adversary cannot distinguish them. The adversary is formalized as any *evaluation context*, defined as follows: An evaluation context C is a context formed with the hole $\{\}$, restrictions $(\nu a)C$, parallel compositions $P \mid C$, and locations $M[C]$. Intuitively, the adversary can listen to messages sent on the network, compute its own messages, and send them. It can also insert the considered process inside locations. The application of context C to process P is denoted by $C\{P\}$. An evaluation context C *closes* P when $C\{P\}$ is closed. We say that a process P *emits on channel* a, and we write $P \Downarrow a$, when $P\,(\rightarrow \cup \equiv)^*\,(\nu b_1)\dots(\nu b_n)(\overline{a}^\star(M).P \mid Q)$ with $a \notin \{b_1, \dots, b_n\}$.

Definition 1. Observational equivalence $\approx$ *is the largest symmetric relation* $\mathcal{R}$ *between closed processes such that* $P \,\mathcal{R}\, Q$ *implies:*

(1) *if* $P \Downarrow a$*, then* $Q \Downarrow a$*;*
(2) *if* $P \rightarrow^* P'$ *then there exists* Q' *such that* $Q \rightarrow^* Q'$ *and* $P' \,\mathcal{R}\, Q'$*;*
(3) $C\{P\} \,\mathcal{R}\, C\{Q\}$ *for all closing evaluation contexts* C*.*

Observational equivalence can be used to define security properties of processes, by showing that the process is observationally equivalent to an ideal process representing the security specification of the system. It can also be used to define secrecy based on the notion of non-interference:

Definition 2. *A process* P preserves the secrecy *of* $fv(P)$ *if and only if for all* σ *and* σ' *substitutions of domain* $fv(P)$*,* $P\sigma \approx P\sigma'$*.*

This definition intuitively expresses that the adversary obtains no information on the value of the variables in $fv(P)$, since processes using different values for these variables are observationally equivalent, i.e. indistinguishable by the adversary. This notion of secrecy makes it possible to detect implicit flows, for instance when the adversary can test whether a secret is equal to some value it has. It is stronger than the perhaps more frequent reachability notion, saying that a value is secret when the adversary cannot reconstruct it. We refer the reader to [1] for a more detailed discussion of the definition of secrecy.

Proving observational equivalence is difficult, because we have to prove equivalence *for all* possible contexts (point (3) of the definition). We define another process equivalence, namely labelled bisimilarity, that avoids the universal quantification on contexts. We show that labelled bisimilarity equals observational equivalence. Such results are frequent in process algebras, but they are also important: they make it possible to replace a proof of observational equivalence with a much simpler proof of labelled bisimilarity.

The definition of labelled bisimilarity proceeds in several steps. First, we define the notion of *agents*. Agents represent the current state of the process as well as the current knowledge of the adversary. An agent A is of the form $(\nu\tilde{a})(\sigma, P)$, where $\tilde{a}$ is a set of names, σ is a substitution from variables to terms, and P is a process, such that $dom(\sigma) \cap (fv(P) \cup \cup_x fv(x\sigma)) = \emptyset$. In the agent $(\nu\tilde{a})(\sigma, P)$, P represents the current state of the process and σ the current knowledge of the adversary, that is, the adversary knows all terms in the image of σ, and it can designate them by using the variables in the domain of σ. The restriction $(\nu\tilde{a})$ indicates that the adversary does not have a priori knowledge of the names in $\tilde{a}$. For example, the agent $(\nu k)(\{\mathbf{encrypt}((a, b), k)/x\}, P)$ represents that the adversary has the term $M = \mathbf{encrypt}((a, b), k)$, but not the key k, so it cannot see that M is an encryption, but it can build (x, a), that is, (M, a), for instance. If it later learns k, with an agent such as $(\nu k)(\{\mathbf{encrypt}((a,b),k)/x, k/y\}, P')$, then it can decrypt M by computing $\mathbf{decrypt}(x, y)$ which is equal to (a, b) taking into account the values of x and y. We define $dom((\nu\tilde{a})(\sigma, P)) = dom(\sigma)$. An agent is closed when the image of σ and the process P do not contain free variables. A process is also an agent, by taking $\tilde{a}$ and $dom(\sigma)$ empty.

Agents are considered equal modulo renaming of bound names and variables. The structural equivalence and reduction of agents extend the ones of processes. More precisely, the structural equivalence of agents is defined as the smallest equivalence relation closed under:

$$(\nu\tilde{a})(\sigma, P) \equiv (\nu\tilde{a})(\sigma', P) \text{ if } \forall x, \Sigma \vdash x\sigma = x\sigma'$$

$$(\nu\tilde{b}, a)(\sigma, P) \equiv (\nu\tilde{b})(\sigma, (\nu a)P) \text{ if } \forall x, a \notin fn(x\sigma)$$

$$P \equiv P' \Rightarrow (\nu\tilde{b})(\sigma, P) \equiv (\nu\tilde{b})(\sigma, P')$$

The reduction of agents is defined by $A \to B$ when $A \equiv (\nu\tilde{b})(\sigma, P)$, $P \to Q$, and $(\nu\tilde{b})(\sigma, Q) \equiv B$.

The labelled operational semantics represents reductions of processes that occur in interaction with an unspecified context. It is defined by the rules of Fig. 3. The label α of the reduction indicates the nature of the interaction with this context: it can be of the form $\overline{M}^{\eta}(x)$ (output of a term on channel M, the term is received in variable x by the context), $M^{\eta}(\lambda M')$ (input of a term M' on channel M), $M''[\overline{M}^{\uparrow}(x)]$, or $M''[M^{\uparrow}(\lambda M')]$ (output and input through a location). The rules defining the labelled reduction $A \xrightarrow{\alpha} A'$ apply only when A is closed, and then A' is closed.

The rule (In) means that the process $a^{\eta}(\lambda x).P$ can receive a term M from the context, then it reduces to $P\{M/x\}$. The rule (Out) means that the process $\overline{a}^{\eta}(M).P$ can output the term M to the context, which receives it in some variable x. The reduced process is P, and the context now has knowledge of the term M, which is represented by the substitution $\{M/x\}$. The rules (Loc In) and (Loc Out) handle inputs and outputs inside a location. Only inputs and outputs to the outside of the location (using $\uparrow$: $a^{\uparrow}(\lambda x)$ and $\overline{a}^{\uparrow}(M)$) are visible to the context, so this is the only case considered in these rules. Intuitively, in these rules, $c[P]$ reduces to $c[A]$, but $c[A]$ is outside the syntax of agents, so we replace it with $c[] \circ A$ which plays the same role but yields a well-formed agent. $A \between Q$, $(\nu a) \circ A$, and $\sigma \circ A$ are used for the same reason in the following rules. The rules (Parallel), (Restriction), and (Agent) handle reductions under evaluation contexts. In the rule (Restriction), note that when $a \in fn(\alpha)$, the context needs to know a to perform the reduction $P \xrightarrow{\alpha} A$, so it cannot perform this reduction on the process $(\nu a)P$. The rule (Agent) is complex. It can be decomposed into three steps: First, from $P \xrightarrow{\alpha} A$, we can infer $(\sigma, P) \xrightarrow{\alpha} \sigma \circ A$, provided that the domain of σ does not overlap with the domain of the substitution of A, that is, with the bound variables of α. Second, we can replace the label α with α', provided that $\Sigma \vdash \alpha\sigma = \alpha'\sigma$, that is, α and α' have the same value given the current knowledge of the adversary. Finally, we add the restriction $(\nu\tilde{a})$, in a

$$a^{\eta}(\lambda x).P \xrightarrow{a^{\eta}(\lambda M)} P\{M/x\} \qquad (M \text{ closed, } \eta \text{ is } \star, \uparrow, \text{ or a name}) \qquad \text{(In)}$$

$$\overline{a}^{\eta}(M).P \xrightarrow{\overline{a}^{\eta}(x)} \{M/x\}, P \qquad (\eta \text{ is } \star, \uparrow, \text{ or a name}) \qquad \text{(Out)}$$

$$\frac{P \xrightarrow{a^{\uparrow}(\lambda M)} A}{c[P] \xrightarrow{c[a^{\uparrow}(\lambda M)]} c[] \circ A} \qquad \text{(Loc In)}$$

$$\frac{P \xrightarrow{\overline{a}^{\uparrow}(x)} A}{c[P] \xrightarrow{c[\overline{a}^{\uparrow}(x)]} c[] \circ A} \qquad \text{(Loc Out)}$$

$$\frac{P \xrightarrow{\alpha} A}{P \mid Q \xrightarrow{\alpha} A \phi Q} \qquad \text{(Parallel)}$$

$$\frac{P \xrightarrow{\alpha} A \qquad a \notin \mathit{fn}(\alpha)}{(\nu a)P \xrightarrow{\alpha} (\nu a) \circ A} \qquad \text{(Restriction)}$$

$$\frac{P \xrightarrow{\alpha} A \qquad bv(\alpha) \cap dom(\sigma) = \emptyset \qquad \Sigma \vdash \alpha\sigma = \alpha'\sigma \qquad \tilde{a} \cap \mathit{fn}(\alpha') = \emptyset}{(\nu\tilde{a})(\sigma, P) \xrightarrow{\alpha'} (\nu\tilde{a}) \circ \sigma \circ A} \qquad \text{(Agent)}$$

$$\frac{A \equiv B \quad B \xrightarrow{\alpha} B' \quad B' \equiv A'}{A \xrightarrow{\alpha} A'} \qquad \text{(Struct)}$$

with

$$(\nu\tilde{a})(\sigma, P) \phi Q = (\nu\tilde{a})(\sigma, P \mid Q\sigma) \text{ if } \mathit{fn}(Q) \cap \tilde{a} = \emptyset$$
$$(\nu b) \circ (\nu\tilde{a})(\sigma, P) = (\nu\tilde{a}, b)(\sigma, P) \text{ if } b \notin \tilde{a}$$
$$\sigma' \circ (\nu\tilde{a})(\sigma, P) = (\nu\tilde{a})(\sigma \cup \sigma'\sigma, P) \text{ if } (\nu\tilde{a})(\sigma, P) \text{ is closed,}$$
$$\text{for all } x, \mathit{fn}(x\sigma') \cap \tilde{a} = \emptyset, \text{ and } dom(\sigma) \cap dom(\sigma') = \emptyset$$
$$c[] \circ (\nu\tilde{a})(\sigma, P) = (\nu\tilde{a})(\sigma, c[P]) \text{ if } c \notin \tilde{a}$$

$$\mathit{fv}(\overline{M}^{\eta}(x)) = \mathit{fv}(M) \cup \mathit{fv}(\eta), \mathit{fv}(M^{\eta}(\lambda M')) = \mathit{fv}(M) \cup \mathit{fv}(\eta) \cup \mathit{fv}(M'),$$
$$\mathit{fv}(M''[\alpha]) = \mathit{fv}(M'') \cup \mathit{fv}(\alpha) \text{ and similarly for the free names } \mathit{fn}.$$
$$bv(\overline{M}^{\eta}(x)) = \{x\}, bv(M^{\eta}(\lambda M')) = \emptyset, bv(M''[\alpha]) = bv(\alpha)$$

Fig. 3. Labelled semantics

way similar to the rule (Restriction). The rule (Struct) allows using structural equivalence to transform agents before reducing them.

We now define the *static equivalence* of agents. Two agents A and B are statically equivalent when the adversary cannot distinguish the terms given by the substitutions of these agents (without executing the processes associated with each of A and B).

Definition 3. *We say that* $(M = N)A$ *if and only if there exist a substitution* σ*, a process* P*, and names* $\tilde{a}$ *such that* $A \equiv (\nu\tilde{a})(\sigma, P)$*,* $\Sigma \vdash M\sigma = N\sigma$*, and* $\tilde{a} \cap (\mathit{fn}(M) \cup \mathit{fn}(N)) = \emptyset$*.*

We say that $\mathrm{isname}(M, A)$ *is true if and only if there exist a substitution* σ*, a process* P*, and names* $\tilde{a}$*,* b *such that* $A \equiv (\nu\tilde{a})(\sigma, P)$*,* $\Sigma \vdash M\sigma = b$*, and* $\tilde{a} \cap fn(M) = \emptyset$*.*

Now, two agents A *and* B *are* statically equivalent *(*$A \approx_{\mathrm{s}} B$*) when* $dom(A) = dom(B)$*, and for all terms* M *and* N*,* $(M = N)A$ *if and only if* $(M = N)B$*, and* $\mathrm{isname}(M, A)$ *if and only if* $\mathrm{isname}(M, B)$*.*

The condition "$(M = N)A$ if and only if $(M = N)B$" takes into account that the adversary can test equality between terms. The property $\mathrm{isname}(M, A)$ means that M is equal to a name, taking into account the values of variables given by the substitution of A. The condition "$\mathrm{isname}(M, A)$ if and only if $\mathrm{isname}(M, B)$" models the adversary's ability to check whether M is a name, by using it as a channel: the process $M^\star(\lambda x).\overline{a}^\star(b) \mid \overline{M}^\star(b)$ emits on channel a if and only if M is equal to a name. Another possibility is to authorize communications when channels and locations are any terms instead of just names. With that semantics, the condition on $\mathrm{isname}(M, A)$ would be removed.

Finally, we can define the labelled bisimilarity:

Definition 4. *Labelled bisimilarity (*$\approx_{\mathrm{l}}$*) is the largest symmetric relation* $\mathcal{R}$ *on closed agents such that* $A \,\mathcal{R}\, B$ *implies:*

(1) $A \approx_{\mathrm{s}} B$*;*
(2) *if* $A \to A'$ *then there exists* B' *such that* $B \to^* B'$ *and* $A' \,\mathcal{R}\, B'$*;*
(3) *if* $A \xrightarrow{\alpha} A'$*, then there exists* B' *such that* $B \to^* \xrightarrow{\alpha} \to^* B'$ *and* $A' \,\mathcal{R}\, B'$*.*

Theorem 1. *Labelled bisimilarity equals observational equivalence:* $P \approx_{\mathrm{l}} Q$ *if and only if* $P \approx Q$*.*

Note that, in higher-order process calculi, the characterization of observational equivalence as labelled bisimilarity is often very difficult [18], because, when the adversary receives a process, the only observation it can do is to execute that process. In our calculus, the adversary can extract data and compare the code of processes (the existence of the function **getdata** is fundamental for that, see Sect. 2), much like it can for any other piece of data. Executing the process does not give more information to the adversary than these operations. Hence, being able to send processes does not complicate the labelled bisimilarity. In contrast, the presence of locations complicates the proof of Th. 1 considerably.

4 Examples

In this section, we present three examples illustrating the features of our calculus.

4.1 Firewalls

Firewalls exist as gateways through which communications with external networks are controlled (or restricted). The concept of a firewall corresponds directly to the concept of a location. Processes within a firewall can only communicate with the external context when the firewall allows it. Such communications require a prior knowledge of the address (name) of the firewall.

A well-known property that characterizes perfectly isolating firewalls is expressed by the *perfect firewall equation* [12]:

Proposition 1. *Let P be a closed process. Then $(\nu c)c[P] \approx 0$.*

This result states that the firewall $(\nu c)c[\{\}]$ is perfectly isolating because its internal behaviour is non-observable due to the hiding effect of (νc). (This can be proved using labelled bisimilarity and noticing that $(\nu c)c[P]$ and all its reductions by the standard reduction relation do not reduce by the labelled reduction.) When P has free variables, it is easy to show, using the above proposition, that $(\nu c)c[P]$ preserves the secrecy of all its free variables, since $(\nu c)c[P\sigma] \approx 0 \approx (\nu c)c[P\sigma']$ for all closed substitutions σ and σ' of domain $fv(P)$.

In practice, a firewall often leaves some channels open for communications by extending the scope of the restriction (νc):

$$R \stackrel{\mathtt{def}}{=} (\nu c)(c[P] \mid\ !a^{\star}(\lambda x).\overline{a}^{c}(x) \mid\ !b^{c}(\lambda x).\overline{b}^{\star}(x)) \mid Q$$

With the two replicated processes aware of the name c, messages may be forwarded over channel a from Q to P, and from P to Q over channel b. Other communications are forbidden by the firewall. Hence, for example:

Proposition 2. *Let $P = \overline{b}^{\uparrow}(y) \mid \overline{d}^{\uparrow}(z)$. Then $R\{M/y\}$ preserves the secrecy of z, and $R\{M/z\}$ does not preserve the secrecy of y.*

Indeed, y is sent to the outside of the firewall on channel b. Another form of firewalls is built by using nested locations:

Proposition 3. $c[c'[P]] \approx 0$.

Two nested locations prevent communications because communications are possible only between locations immediately inside one another. We can allow some communications by extending the scope of the location c, to include some relay processes. For example, we can model a computer as a location c with processes running inside it as locations $c_1, \ldots, c_n$ by the process:

$$c[c_1[P_1] \mid \ldots \mid c_n[P_n] \mid P]$$

The process P models the operating system. Then distinct processes can communicate with the network outside the computer and with each other only through the operating system P acting as the intermediary.

We can also model a Local Area Network (LAN) as a location c, containing several locations $c_1, \ldots, c_n$, each representing a computer, by the same process. The process P the models the routing of messages inside the LAN and the communications with the outside. (Realistically, P should simply forward all communications from a location inside the LAN to another location inside the LAN, with only some communications forwarded to the outside.)

4.2 Applets

The **pack** and **exec** primitives for packing processes and extracting them from messages allow for the modelling of applets. For example, consider the following sequence of messages exchanged between a client C and a server S:

$$\begin{aligned} &C \to S : \text{“Send me your applet”} \\ &S \to C : \{\mathbf{pack}(\vec{M}, \lambda\vec{y}.P)\}_{sk_S} \end{aligned}$$

where digital signatures, used in the second message, require the function **sign** to sign a message, **checksign** to check it, **extract** to extract the message from a signed message, and **pk** to create a public key from its secret part. These are defined by the equations:

$$\begin{aligned} \mathbf{checksign}(\mathbf{sign}(M, sk), \mathbf{pk}(sk)) &= \mathbf{True} \\ \mathbf{checksign}(M', \mathbf{pk}(sk)) &= \mathbf{False} \quad \text{if } M' \neq \mathbf{sign}(M, sk) \text{ for all } M \\ \mathbf{extract}(\mathbf{sign}(M, sk)) &= M \end{aligned}$$

The system can then be represented by the following processes:

$$R \stackrel{\mathsf{def}}{=} (\nu sk_S) let\ pk_S = \mathbf{pk}(sk_S)\ in\ (\overline{c}^\star(pk_S) \mid C \mid S)$$
$$C \stackrel{\mathsf{def}}{=} \overline{c}^\star(\mathbf{sndapp}).c^\star(\lambda x).if\ \ \mathbf{checksign}(x, pk_S) = \mathbf{True}\ then\ \mathbf{exec}(\mathbf{extract}(x))$$
$$S \stackrel{\mathsf{def}}{=} !c^\star(\lambda x).if\ \ x = \mathbf{sndapp}\ \ then\ \overline{c}^\star(\mathbf{sign}(\mathbf{pack}(\vec{M}, \lambda\vec{y}.P), sk_S))$$

The process R first creates a new secret key for the server sk_S, then computes the corresponding public key pk_S, and publishes this key by $\overline{c}^\star(pk_S)$, so that the adversary can have it. Finally, it executes the client and server processes (C and S) according to the message sequence above.

The client is assured of the origin of the applet when it succeeds in verifying the signature using **checksign** (assuming the public key pk_S is validly bound to the server S at the time when **checksign** is called). It then executes the applet. Otherwise, if the applet is not signed, or it is signed with a key different from sk_S, then the client terminates.

One may arrive at the following result, which shows that the only applet that can be executed by R is $\mathbf{pack}(\vec{M}, \lambda\vec{y}.P)$:

Proposition 4. *Let R' be obtained from R by substituting “if* $\mathbf{extract}(x) = \mathbf{pack}(\vec{M}, \lambda\vec{y}.P)$ *then* $P\{\vec{M}/\vec{y}\}$*” for “*$\mathbf{exec}(\mathbf{extract}(x))$*” in C. Then $R \approx R'$.*

The key lemma to prove this result is to show that the only packed process that can be the parameter of **exec** is the one signed by the server.

Other protocols to obtain applets are possible. We have also modelled an example in which an applet is sent encrypted, using a simple public-key protocol. We could also model an SSL communication on which an applet is sent.

Applets that are communicated by secure means (encryption or signatures) as shown in the previous example are often trusted to a certain degree, depending on the level of trust associated with the public key of the server pk_S. However, applets received through insecure communications (non-trusted applets) are run in special *sandboxes* that limit their capabilities. For example, such applets are prevented from accessing the local file system or contacting network addresses (except the address from which the applet originated).

Consider again the client C in the signed applet example above. This time it is enhanced with a sandbox $sb[\ldots]$, within which applets not originating from the server S are run. On the other hand, trusted applets are given more privileges by running them outside $sb[\ldots]$:

$$C \stackrel{\mathbf{def}}{=} (\overline{c}^\star(\mathbf{sndapp}).c^\star(\lambda x).\mathit{if}\ \mathbf{checksign}(x, pk_S) = \mathbf{True}\ \mathit{then}\ \mathbf{exec}(\mathbf{extract}(x)) \\ \mathit{else}\ (\nu sb)(sb[\mathbf{exec}(\mathbf{extract}(x))] \mid P)) \quad \mid Q$$

Process P models the Java standard library. Its communications with the applet correspond to method calls and returns. The process P can communicate with the outside of the sandbox, and for instance send messages coming from the applet on the network (perhaps after examining them for security). Process Q, on the other hand, is protected from the applet by the fact that it falls outside the scope of the sandbox. Prop. 4 can be applied to the first occurrence of $\mathbf{exec}(\mathbf{extract}(x))$ in C, and the sandbox prevents direct communications between the applet and the outside world: all these communications must go through P, since one needs to know sb to communicate with the interior of the sandbox. (Obviously, we assume that P is disciplined so as not to reveal the name sb.)

4.3 Certified Email Protocol

We consider here a cryptographic protocol whose implementation relies on a Java applet: the email protocol of [5]. In this protocol, a sender S sends an email message to a receiver R, such that R has the message if and only if S has a receipt. The protocol also involves a trusted third party TTP. In a slightly simplified version, the protocol runs as follows:

Message 1. $\mathsf{S} \rightarrow \mathsf{R}$: "This is a certified email.
Please read it in a Java enabled browser"
`<APPLET CODE="https://ttp.com/applet.jar"`
`PARAM=`$(\mathsf{cleartext}, \mathsf{em}, \mathsf{S2TTP})$`>`
Message 2. $\mathsf{R} \rightarrow \mathsf{TTP}$: get `https://ttp.com/applet.jar`
Message 3. $\mathsf{TTP} \rightarrow \mathsf{R}$: `applet.jar`
Message 4. $\mathsf{R} \rightarrow \mathsf{TTP}$: $\mathsf{S2TTP}$, "owner of RPwd wants key for h"
Message 5. $\mathsf{TTP} \rightarrow \mathsf{R}$: "try k for h"
Message 6. $\mathsf{TTP} \rightarrow \mathsf{S}$: $\mathbf{sign}$("I have released the key for $\mathsf{S2TTP}$ to R", $\mathsf{sk}_{\mathsf{TTP}}$)

The sender S wants to send to R the message m. It creates a fresh key k, encrypts m with k, obtaining $\mathsf{em} = \mathbf{encrypt}(\mathsf{m}, \mathsf{k})$. It also computes $\mathsf{h} = \mathbf{H}((\mathsf{cleartext}, \mathsf{em}))$ and $\mathsf{S2TTP} = \mathbf{pencrypt}((\mathsf{S}, \text{"give } \mathsf{k} \text{ to } \mathsf{R} \text{ for } \mathsf{h}\text{"}), \mathsf{pk}_{\mathsf{TTP}})$, where $\mathbf{pencrypt}$ is public-key encryption and **H** is a one-way hash function. Then it sends to R an html email (message 1) containing a header explaining how to read the email, as well as a reference to an applet `https://ttp.com/applet.jar` taking as parameters (cleartext, em, S2TTP), where cleartext is an explanation of the contents of the certified e-mail. The reader's browser downloads the applet on a secure SSL connection (messages 2 and 3) and executes it with the indicated parameters. The applet displays cleartext to the receiver, and asks him for his password RPwd. If R decides to read the email, it enters its password RPwd and the applet sends message 4 to TTP on a secure channel built by a combination of public-key and shared-key encryption. The value of h in this message is recomputed by the applet from cleartext and em. TTP replies to R with message 5 on the same secure channel as message 4. Upon receipt of message 5, the applet decrypts em with k, and displays m. TTP also sends the receipt (message 6) to S. The delivery of messages 5 and 6 is guaranteed. We have formally modeled this protocol in our calculus, by combining the model of [2] (which omitted the applet) and a model of the applet along the same lines as the previous example. The resulting model is 229 lines long (including comments) so we do not detail it here due to the lack of space.

When one assumes that the applet behaves as expected, the protocol satisfies its security properties [2, 7]. However, when one takes into account the downloading of the applet, one finds the following attack, which was mentioned to us by the authors of the protocol: in message 1, S sends the address of an applet on its own web site. This applet behaves as expected, except that it never displays the clear message to R. Because of the security constraints on applets, the applet can contact only the web site it comes from. However, this web site will simply forward the messages to TTP. Then the protocol runs as expected, the sender gets its receipt in message 6, but the receiver never has the clear message because the applet does not display it. This contradicts the main security property of the protocol.

One can see several corrections to this problem, so that the applet can come only from TTP. The easiest one would be for the receiver to manually check the address of the applet. Since the address contains `https`, the applet is downloaded using SSL, so we have the guarantee that the obtained applet is the expected one, as mentioned in the previous example. A result similar to Prop. 4 formalizes this guarantee. Then we can use the security properties of the protocol without applet to show the security of the protocol with applet. We could also require that the applet be signed by TTP. However, browsers accept to run unsigned applets, so the receiver should have a specific piece of software that checks that. In that case, it is probably simpler not to use an applet at all: R should simply have downloaded the software that executes the protocol.

In this example, our calculus makes it possible to formally model the protocol including the applet and a combination of mobility and cryptography (the applet is sent on an SSL connection and also performs cryptographic operations itself). An attack appears in this model while it does not in usual models of protocols that cannot represent applets. Indeed, this attack relies crucially on furnishing a malicious applet to the receiver. We can also prove the correctness of corrected versions of the protocol.

5 Comparison with Other Process Calculi

Models of cryptography Several variants of the spi calculus [6] have been proposed for modelling cryptography. Here, we focus on the most powerful of these variants, the applied pi calculus [3]. We extend this calculus by adding locations and packed processes. There are other relatively important differences between our calculus and the applied pi calculus. We do not introduce floating substitutions and restriction on variables in the calculus, since these are useful only for the labelled bisimilarity. We prefer keeping the calculus as standard and simple as possible, even if the proof of Th. 1 requires a few more technical lemmas. We also remove the "open" rules in the labelled bisimilarity, and do not resort to a sort system to prevent channel names from being manipulated by functions. To compensate this point, our static equivalence takes into account that the adversary can test whether a term is a name by using it as a channel.

Models of location-based mobility One of the most well-known models of mobility is the ambient calculus [12]. In this calculus, locations can move according to their own will and open other ambients. It is very difficult to restrict these movements. This makes the modelling of security properties difficult. Several variants have been designed to solve this problem: the safe ambients [16], in which each movement or opening of an ambient must be authorized by a co-capability, the secure safe ambients [9], which are a variant of safe ambients with a security type system, and the boxed ambients [10] in which the opening of ambients is removed and replaced with non-local communications between ambients. We use the communication mechanism of boxed ambients, but not the ambient mobility model: we believe that objective mobility (with processes sent as messages) is closer to what happens most often in networks. In contrast, ambient calculi are probably better at modelling situations in which human beings move security devices, for example.

Boxed ambients and cryptography have already been combined in [11]. The approach differs from our work in many aspects: [11] considers only shared-key cryptography; it represents messages as moving ambients, which, we think, is less natural than considering them as sent messages; it gives a type system for proving secrecy, whereas we focus on the proof of observational equivalence.

The seal calculus [20] provides strong objective mobility: processes are sent on the network by specific input and output constructs. We focus on weak mobility, and simplify the calculus by using the ordinary input and output instructions to manipulate packed processes. In the original version [20], the seal calculus used an **open** construct, combined to a local input or output to create non-local communications. In a recent draft, G. Castagna, F. Nardelli, and J. Vitek have adopted a simpler communication mechanism similar to the one of boxed ambients, which we adopt.

The Dπ-calculus of [17] includes a notion of migration of the current location (as in ambients) but also features such as halted and running locations, that we omit from our calculus for simplicity. In the distributed π-calculus of [19], each subterm is explicitly located, instead of having locations contain a process, and communications can take place between different locations without constraints.

The join calculus The previous models are all based on the pi calculus. Another process calculus that can be used for mobility is the join calculus [13]. It has the same expressive power as the pi calculus, but uses asymmetric channels with one receiver instead of symmetric channels. (One type of channel can encode the other.) It has also been extended to cryptography, similarly to the spi calculus [4]. The distributed join calculus [15] considers a tree of nested locations, with subjective mobility (but no cryptography).

6 Conclusion

In this paper, we have presented crypto-loc, a process calculus combining the concepts of cryptography, locations, and weak code mobility. We have also defined a theory of process equivalence, and showed that observational equivalence equals labelled bisimilarity. The theory was used in defining and proving security properties.

An interesting area for further research would involve extending automatic static analyses techniques that have already been developed for cryptographic protocols to this calculus. Indeed, the presence of locations, and of communication and cryptographic operations on code raises new difficulties for static analysis.

Acknowledgments

We would like to thank Cédric Fournet and Jan Vitek for very helpful comments on this work.

References

1. M. Abadi. Security Protocols and their Properties. In *Foundations of Secure Computation*, NATO Science Series, pages 39–60. IOS Press, 2000. Volume for the 20th International Summer School on Foundations of Secure Computation, held in Marktoberdorf, Germany (1999).
2. M. Abadi and B. Blanchet. Computer-Assisted Verification of a Protocol for Certified Email. In *Static Analysis, 10th International Symposium (SAS'03)*, volume 2694 of *LNCS*, pages 316–335. Springer, June 2003.
3. M. Abadi and C. Fournet. Mobile Values, New Names, and Secure Communication. In *28th ACM Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115. ACM, Jan. 2001.
4. M. Abadi, C. Fournet, and G. Gonthier. Secure Implementation of Channel Abstractions. *Information and Computation*, 174(1):37–83, Apr. 2002.

5. M. Abadi, N. Glew, B. Horne, and B. Pinkas. Certified Email with a Light On-line Trusted Third Party: Design and Implementation. In *Proceedings of the Eleventh International World Wide Web Conference*, pages 387–395. ACM, May 2002.
6. M. Abadi and A. D. Gordon. A Calculus for Cryptographic Protocols: The Spi Calculus. *Information and Computation*, 148(1):1–70, Jan. 1999.
7. G. Bella, C. Longo, and L. C. Paulson. Verifying Second-Level Security Protocols. In *Theorem Proving in Higher Order Logics (TPHOLs'03)*, LNCS. Springer, Sept. 2003. To appear.
8. L. Bettini and R. D. Nicola. Translating Strong Mobility into Weak Mobility. In *Mobile Agents, 5th International Conference, MA'01*, volume 2240 of *LNCS*, pages 182–197. Springer, Dec. 2001.
9. M. Bugliesi and G. Castagna. Secure Safe Ambients. In *28th ACM Symposium on Principles of Programming Languages (POPL'01)*, pages 222–235. ACM, Jan. 2001.
10. M. Bugliesi, G. Castagna, and S. Crafa. Boxed Ambients. In *Theoretical Aspects of Computer Software, 4th International Symposium, TACS'01*, volume 2215 of *LNCS*, pages 38–63. Springer, Oct. 2001.
11. M. Bugliesi, S. Crafa, A. Prelić, and V. Sassone. Secrecy in Untrusted Networks. In *13th International Colloquium on Automata, Languages and Programming (ICALP'03)*, volume 2719 of *LNCS*, pages 969–983. Springer, July 2003.
12. L. Cardelli and A. D. Gordon. Mobile Ambients. In *Foundations of Software Science and Computation Structures, First International Conference (FoSSaCS '98)*, volume 1378 of *LNCS*, pages 140–155. Springer, 1998.
13. C. Fournet and G. Gonthier. The Reflexive Chemical Abstract Machine and the Join-Calculus. In *23rd ACM Symposium on Principles of Programming Languages (POPL'96)*, pages 372–385, Jan. 1996.
14. C. Fournet and G. Gonthier. A hierarchy of equivalences for asynchronous calculi. In *Proceedings of the 25th International Colloquium on Automata, Languages and Programming (ICALP'98)*, volume 1443 of *LNCS*, pages 844–855. Springer, July 1998.
15. C. Fournet, G. Gonthier, J.-J. Lévy, L. Maranget, and D. Rémy. A Calculus of Mobile Agents. In *7th International Conference on Concurrency Theory (CONCUR'96)*, volume 1119 of *LNCS*, pages 406–421. Springer, Aug. 1996.
16. F. Levi and D. Sangiorgi. Controlling Interference in Ambients. In *27th ACM Symposium on Principles of Programming Languages (POPL'00)*, pages 352–364. ACM, Jan. 2000.
17. J. Riely and M. Hennessy. A Typed Language for Distributed Mobile Processes. In *25th ACM Symposium on Principles of Programming Languages (POPL'98)*, pages 378–390. ACM, Jan. 1998.
18. D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms.* PhD thesis, University of Edinburgh, 1992.
19. P. Sewell. Global/Local Subtyping and Capability Inference for a Distributed π-calculus. In *Automata, Languages and Programming, 25th International Colloquium, ICALP'98*, volume 1443 of *LNCS*, pages 695–706. Springer, July 1998.
20. J. Vitek and G. Gastagna. Seal: A Framework for Secure Mobile Computations. In *Internet Programming Languages, ICCL'98 Workshop*, volume 1686 of *LNCS*, pages 47–77. Springer, May 1999.