# Spi Calculus Translated to $\pi$-Calculus Preserving May-Tests *

Michael Baldamus Joachim Parrow Björn Victor

Department of Information Technology, Uppsala University, Sweden
{*Michael.Baldamus,Joachim.Parrow,Bjorn.Victor*}*@it.uu.se*

## Abstract

*We present a concise and natural encoding of the spi-calculus into the more basic $\pi$-calculus and establish its correctness with respect to a formal notion of testing. This is particularly relevant for security protocols modelled in spi since the tests can be viewed as adversaries. The translation has been implemented in a prototype tool. As a consequence, protocols can be described in the spi calculus and analysed with the emerging flora of tools already available for $\pi$. The translation also entails a more detailed operational understanding of spi since high level constructs like encryption are encoded in a well known lower level. The formal correctness proof is nontrivial and interesting in its own; so called context bisimulations and new techniques for compositionality make the proof simpler and more concise.*

## 1 Introduction

The current proliferation of computer communication services and technologies is accompanied by an equally bewildering plethora of different formal description techniques. There are many different families with different purposes. Some of them, like the $\pi$-calculus [16], aim at a basic formalism with few and low-level primitives, applicable as a springboard for more high-level and specialised techniques. As an example the spi calculus [2, 4] is developed as an extension of it with high-level primitives for among other things encryption and decryption. This makes the spi calculus especially appropriate to describe authentication protocols and services. It is natural to ask if the added complexity of the spi calculus is necessary in a formal sense, or if there is a natural encoding in terms of the more basic $\pi$-calculus. The contribution of this paper is to exhibit such an encoding together with new proof techniques to establish its properties.

Our encoding is surprisingly natural and concise. The main idea is to represent spi calculus terms as objects with a number of predefined methods. Encryption corresponds to creating an object with a special method for decryption; the object will perform this only when the correct encryption key is presented. This entails a distributed view of encryptions and avoids a global repository of cleartexts and keys in the formal model. In other words, we establish that cryptography can be encoded into the $\pi$-calculus along standard lines of encoding functions and data structures.

The spi calculus is expressively powerful enough to model not only a large class of authentication protocols but also a large class of potential adversaries trying to break them. Thus a formal notion of correctness of a protocol is that it satisfies tests which themselves are spi calculus terms. Granted, these tests will not be able to model attacks on the fundamental assumptions (for example that ciphertexts can never be decrypted without access to the encryption key and that it is impossible to interfere with the execution of principals), but they do capture the fact that a protocol is correct under those assumptions. Our main technical result of the encoding is that it is faithful to such tests in a formal sense. Let $P$ be a protocol and $E$ a test, both formulated in the spi calculus, and let $[\![\cdot]\!]$ be our encoding from the spi calculus to the $\pi$-calculus. We prove that

$$P \text{ satisfies } E \text{ iff } [\![P]\!] \text{ satisfies } [\![E]\!].$$

Thus, analyses carried out on the encodings are relevant for the original spi calculus descriptions. As expected the formal proof of this is nontrivial. We develop techniques based on so called context bisimulations previously defined in higher-order calculi, and new techniques for compositionality make the proof simpler and more concise. To our knowledge, no equally complicated term-enriched variant of the $\pi$-calculus has been analysed in this way, with a translation into $\pi$ that is formally proved adequate.

One implication of our work is that automated analysis of spi calculus descriptions can be conducted by first translating them to the $\pi$-calculus. There is today an emerging flora of automatic tools for the $\pi$-calculus, and we have implemented the translation in a prototype that works with some of these tools. This opens the opportunity to formally

*Work supported by European Union project PROFUNDIS, Contract No. IST-2001-33100.

connect the efforts on tools and algorithms related to the calculi. Of course much work remains to determine exactly what analyses can be performed effectively and where the translational approach holds advantages. Existing tools for calculi like spi (see e.g. [7, 9, 5]) can verify individual properties, often by analysing traces. There has been progress also with regard to the more global, equivalence-based approach (see [10] for a good overview), but to date no tool seems to exist for that. As equivalence-checking tools exist for the $\pi$-calculus, we hope that our spi-to-$\pi$ encoding might ultimately help in developing such tools.

It has to be pointed out that our translation only preserves may-*tests* in the way indicated above but not may-*testing equivalence*. To be more precise, an easy corollary of our result is that if the $\pi$-calculus agents $[\![P]\!]$ and $[\![Q]\!]$ are may-equivalent (meaning they pass the same tests) then also the spi-calculus agents $P$ and $Q$ are may-equivalent. However the converse does not hold: may-equivalence on the spi side does not imply may-equivalence on the $\pi$-side (cf. Section 5). In other words, our translation does not solve the well-known full abstraction problem for may-testing equivalence over the spi calculus.

In the next section we give the formal syntax and semantics of the spi- and $\pi$-calculi; the paper is formally self contained although a reader completely unfamiliar with these calculi will probably have difficulty in appreciating the paper. In Section 3 we present and explain the encoding in detail, and the following section contains a detailed sketch of the main technical result. The final section comments on related and future work.

## 2 The $\pi$- and the Spi-Calculus

In this section we present the syntax and operational semantics of the versions of the $\pi$-calculus and spi calculus used.

### 2.1 The Polyadic $\pi$-Calculus

We assume an infinite enumerable set $\mathcal{N}_\pi$ of names, ranged over by $a, b, \ldots$. Names represent communication channels, and are also the values passed in communication.

$\pi$-calculus agents are ranged over by $P$, $Q$, $R$, and are formed by the following grammar:

$$\begin{array}{rcl} P & ::= & \mathbf{0} \mid \alpha.P \mid P \mid Q \mid P + Q \\ & & \mid (\nu\, a)\, P \mid [a = b]\, P \mid\ !\, P \end{array}$$

$\mathbf{0}$ is the deadlocked process which can perform no actions. $\alpha.P$ can perform the action $\alpha$ and continue as $P$; $P \mid Q$ acts as $P$ and $Q$ in parallel; $(\nu\, a)\, P$ binds the name $a$ in $P$ and no process outside $P$ knows about $a$ unless $P$ reveals it by communication; $[a = b]\, P$ can proceed as $P$ only if $a$ and $b$ are the same; $!\, P$ represents any number of copies of $P$ in parallel. We write $(\nu\, \widetilde{a})\, P$ for $(\nu\, a_1) \cdots (\nu\, a_k)\, P$ where $\widetilde{a}$ is the vector $a_1 \cdots a_k$, $k \geq 0$.

Prefixes, ranged over by $\alpha$, are outputs $\overline{a}\langle\widetilde{b}\rangle$ which send the vector of names $\widetilde{b}$ on the channel $a$, and inputs $a(\widetilde{b})$, where all names in $\widetilde{b}$ are distinct, which receive names on the channel $a$ and substitute them for $\widetilde{b}$ in the prefixed agent.

The $\nu$ operator and the input prefix are name binders. We write $\mathrm{bn}(P)$ and $\mathrm{fn}(P)$ for the bound and free names of $P$, respectively. We assume, from now on, that all bound names are distinct from each other and from all free names. Also, we identify agents that can be alpha-converted into each other. The result of substituting a name $a$ for (all free occurrences of) a name $b$ in an agent $P$ is denoted by $P[b := a]$, with the expected extension to equal-length vectors $\widetilde{a}$ and $\widetilde{b}$ of names, where all names in $\widetilde{b}$ are mutually distinct.

The operational semantics is given in Table 1 (cf. also [19]), where transition labels, ranged over by $\mu$, are inputs $a(\widetilde{b})$, outputs $(\nu\, \widetilde{b}')\, \overline{a}\langle\widetilde{b}\rangle$ (where $\widetilde{b}' \subseteq \widetilde{b}$ are extruded by the output), and the internal action $\tau$.

### 2.2 The Spi Calculus

The spi calculus extends the $\pi$-calculus with primitives for encryption, decryption and hashcodes, and pairs and the natural numbers as basic datatypes. Encryption is assumed to be perfect, i.e., the possibility to break a cipher by cryptoanalysis, chance, or brute force is ignored. The focus is not on cryptographic algorithms but on protocols using them.

We assume infinite enumerable sets $\mathcal{N}_{spi}$, ranged over by $m$, $n$, ..., and $\mathcal{V}_{spi}$, ranged over by $x$, $y$, .... $\mathcal{N}_{spi}$ represents communication channels and atomic encryption keys; $\mathcal{V}_{spi}$ represents variables to be instantiated by communication.

In the spi calculus, the values sent and received are *terms*, ranged over by $M$, $N$, $K$. These are defined by

| | | |
|---|---|---|
| $M ::= n \mid x$ | | name, variable |
| $\mid 0 \mid \mathrm{suc}(M)$ | | zero, successor |
| $\mid (M, N)$ | | pair |
| $\mid \{M\}_N$ | | symmetric encryption |
| $\mid \{\!\mid M \mid\!\}_N$ | | asymmetric encryption |
| $\mid M^+ \mid M^-$ | | private, public key |
| $\mid \mathrm{hashc}(M)$ | | hashcode |

and the spi calculus agents $\mathcal{A}_{spi}$, ranged over by $P$, $Q$, $R$, are defined by

| | |
|---|---|
| $P ::= \mathbf{0} \mid \alpha.P \mid P \mid Q \mid (\nu\, n)\, P$ | |
| $\mid [M\ \mathbf{is}\ N]\, P \mid\ !\, P$ | |
| $\mid \mathbf{let}\ (x, y) = M\ \mathbf{in}\ P$ | (pair splitting) |
| $\mid \mathbf{case}\ M\ \mathbf{of}\ 0 : P\ \mathrm{suc}(x) : Q$ | (integer case) |
| $\mid \mathbf{case}\ M\ \mathbf{of}\ [x]_N\ \mathbf{in}\ P$ | (decryption) |

$$P + \mathbf{0} \equiv P \quad P + Q \equiv Q + P \quad (P + Q) + R \equiv P + (Q + R)$$
$$P \mid \mathbf{0} \equiv P \quad P \mid Q \equiv Q \mid P \quad (P \mid Q) \mid R \equiv P \mid (Q \mid R)$$
$$(\nu\, a)\, P \equiv P \text{ if } a \notin \mathrm{fn}(P) \quad (\nu\, a)\,(\nu\, b)\, P \equiv (\nu\, b)\,(\nu\, a)\, P \quad (\nu\, a)(P \mid Q) \equiv ((\nu\, a)\, P) \mid Q \text{ if } a \notin \mathrm{fn}(Q)$$

$$\frac{P \equiv Q \xrightarrow{\mu} Q \equiv P'}{P \xrightarrow{\mu} P'} \quad a(\widetilde{b}).P \xrightarrow{a(\widetilde{b})} P \quad \overline{a}\langle\widetilde{b}\rangle.P \xrightarrow{\overline{a}\langle\widetilde{b}\rangle} P \quad \frac{P \xrightarrow{\mu} P'}{P + Q \xrightarrow{\mu} P'} \quad \frac{P \xrightarrow{\mu} P'}{P \mid Q \xrightarrow{\mu} P' \mid Q}\ \mathrm{bn}(\mu) \cap \mathrm{fn}(Q) = \emptyset$$

$$\frac{P \xrightarrow{(\nu\, \widetilde{b}')\,\overline{a}\langle\widetilde{b}\rangle} P' \quad Q \xrightarrow{a(\widetilde{c})} Q'}{P \mid Q \xrightarrow{\tau} (\nu\, \widetilde{b}')(P' \mid Q'[\widetilde{c} := \widetilde{b}])}\ |\widetilde{b}| = |\widetilde{c}|,\ \widetilde{b}' \cap \mathrm{fn}(Q) = \emptyset \quad \frac{P \xrightarrow{\mu} P'}{(\nu\, a)\, P \xrightarrow{\mu} (\nu\, a)\, P'}\ a \notin \mathrm{n}(\mu) \quad \frac{P \xrightarrow{(\nu\, \widetilde{b}')\,\overline{a}\langle\widetilde{b}\rangle} P' \rightarrow}{(\nu\, c)\, P \xrightarrow{(\nu\, c, \widetilde{b}')\,\overline{a}\langle\widetilde{b}\rangle} P' \rightarrow}\ a \neq c,\ c \in \widetilde{b} \setminus \widetilde{b}'$$

$$\frac{P \xrightarrow{\mu} P'}{[a = a]\, P \xrightarrow{\mu} P'} \quad \frac{P \xrightarrow{\mu} P'}{!\, P \xrightarrow{\mu} P' \mid !\, P}\ \mathrm{bn}(\mu) \cap \mathrm{fn}(P) = \emptyset \quad \frac{P \xrightarrow{(\nu\, \widetilde{c}')\,\overline{a}\langle\widetilde{b}\rangle} P_1' \rightarrow \quad P \xrightarrow{a(\widetilde{c})} P_2'}{!\, P \xrightarrow{\tau} ((\nu\, \widetilde{b}')(P_1' \mid P_2'[\widetilde{c} := \widetilde{b}])) \mid !\, P}\ |\widetilde{b}| = |\widetilde{c}|,\ \widetilde{b}' \cap \mathrm{fn}(P) = \emptyset$$

**Table 1. Structural congruence and SOS clauses for late semantics for $\pi$-calculus.**

The first few agent constructions are familiar from the $\pi$-calculus. The **let** and **case** constructions are used for projection: pair splitting binds $x$ and $y$, which must be distinct, to the first and second component of $M$ and proceeds as $P$ (if $M$ is a pair); integer case proceeds as $P$ if $M$ is 0 or binds $x$ to its predecessor and proceeds as $Q$ (if $M$ is a positive natural number); decryption binds $x$ and proceeds as $P$ if $M$ is a term encrypted with a key matching $N$.

Prefixes in spi calculus are outputs, $\overline{N}\langle M\rangle$, which sends the term $M$ on the channel $N$, and inputs, $N(x)$, which receives a term on $N$ and substitutes it for $x$ in the prefixed agent. Here the term $N$ must evaluate to a name of a communication channel.

Names are bound by the $\nu$ operator, while variables are bound by input prefixes and **case** constructions. We extend $\mathrm{bn}(P)$ and $\mathrm{fn}(P)$ as expected, and write $\mathrm{fv}(P)$ for the free variables of $P$. We still assume that all bound names and variables are distinct from each other and from all free names and variables, and we identify agents that can be alpha-converted into each other. Substitutions are extended to terms for variables, $P[\widetilde{x} := \widetilde{M}]$.

The operational semantics of the closed agents of our spi calculus are given by the structural axioms and rules listed in Table 2 (cf. also [4]).

This semantics is modelled on the one for $\pi$-calculus in Table 1, where transition labels, ranged over by $\mu$, are $n(x)$, $(\nu\, m)\,\overline{n}\langle M\rangle$ and $\tau$. The only essential new ingredient is that an auxiliary commitment relation is used, intuitively representing evaluation of terms.

## 3 The Translation

The main idea of the translation from the spi calculus to the $\pi$-calculus is to represent spi calculus terms as objects with a number of predefined methods. Encryption, e.g., corresponds to creating an object with a special method for decryption; the object will perform this only when the correct encryption key is presented.

The main complication turns out to lie in representing equality. In the spi calculus checking equality of terms is a primitive operation whereas the $\pi$-calculus only has equality of atomic names. Therefore each term needs a particular method to determine if another term is equal; such methods actually make up the bulk of the encoding.

In the translation $\mathcal{N}_{spi}$ and $\mathcal{V}_{spi}$ are both represented by $\mathcal{N}_\pi$. Spi-calculus agents and terms are translated to $\pi$-calculus agents where some names (called *reserved* names) are used to signal operations on the encodings of terms. Let $P$ be a spi-calculus agent containing a spi-calculus term $M$. In the translation $[\![P]\!]$ of $P$ there will then occur, for some name $\ell$ restricted in $[\![P]\!]$, the translation $[\![M]\!]_\ell$ of $M$ located at $\ell$, meaning that other parts of $[\![P]\!]$ will be able to access the translation of $M$ by using the link $\ell$. Over this link a challenge-response protocol is used to perform operations on the term. Three names are passed to the term encoding in the challenge:

- $c$ representing the operation to perform, e.g., **id** to get the identity of an encoded name, or **match** to check for equality to another term.
- $m$ representing a parameter to the operation, e.g., the location of the term to compare with in a **match** operation;
- $r$ being the name of the channel to send a response or result on. The type of response depends on the operation, e.g., for **id** the name being encoded is passed over $r$, while for **match** no object is needed – the synchronization on $r$ in itself means the terms did match.

The names of operations are reserved, i.e. we assume they do not appear anywhere in the source term or agent.

$$P \mid \mathbf{0} \equiv P \quad P \mid Q \equiv Q \mid P \quad (P \mid Q) \mid R \equiv P \mid (Q \mid R)$$
$$(\nu\, a)\, P \equiv P \text{ if } a \notin \mathrm{fn}(P) \quad (\nu\, a)\,(\nu\, b)\, P \equiv (\nu\, b)\,(\nu\, a)\, P \quad (\nu\, a)(P \mid Q) \equiv ((\nu\, a)\, P) \mid Q \text{ if } a \notin \mathrm{fn}(Q)$$

$[M \textbf{ is } M]\, P > P$
$\textbf{let } (x, y) = (M, N) \textbf{ in } P > P[(x, y) := (M, N)]$
$\textbf{case } 0 \textbf{ of } 0 : P \ \mathrm{suc}(x) : Q > P$
$\textbf{case } \mathrm{suc}(M) \textbf{ of } 0 : P \ \mathrm{suc}(x) : Q > Q[x := M]$
$\textbf{case } \{M\}_N \textbf{ of } [x]_N \textbf{ in } P > P[x := M]$
$\textbf{case } \{|M|\}_N \textbf{ of } [x]_{N^{-1}} \textbf{ in } P > P[x := M]$
$N = K^+$ or $N = K^-$ for some $K$,
where $(K^-)^{-1} = K^+$ and $(K^+)^{-1} = K^-$

$$\frac{P \equiv Q \xrightarrow{\mu} Q \equiv P'}{P \xrightarrow{\mu} P'} \quad \frac{P > P' \quad P' \xrightarrow{\mu} P''}{P \xrightarrow{\mu} P''} \quad m(x).P \xrightarrow{m(x)} P \quad \overline{m}\langle M\rangle.P \xrightarrow{\overline{m}\langle M\rangle} P \quad \frac{P \xrightarrow{(\nu\, \widetilde{n})\,\overline{m}\langle M\rangle} P' \quad Q \xrightarrow{m(x)} Q'}{P \mid Q \xrightarrow{\tau} (\nu\, \widetilde{n})(P' \mid Q'[x := M])} \ \widetilde{n} \cap \mathrm{fn}(Q) = \emptyset$$

$$\frac{P \xrightarrow{\mu} P'}{P \mid Q \xrightarrow{\mu} P' \mid Q} \quad \frac{P \xrightarrow{\mu} P'}{(\nu\, m)\, P \xrightarrow{\mu} (\nu\, m)\, P'} \ m \notin \mathrm{n}(\mu) \quad \frac{P \xrightarrow{(\nu\, \widetilde{n})\,\overline{m}\langle M\rangle} P' \quad m \neq k}{(\nu\, k)\, P \xrightarrow{(\nu\, k,\, \widetilde{n})\,\overline{m}\langle M\rangle} P'} \ k \notin \mathrm{fn}(M) \setminus n'$$

$$\frac{P \xrightarrow{\mu} P'}{!\, P \xrightarrow{\mu} P' \mid !\, P} \ \mathrm{bn}(\mu) \cap \mathrm{fn}(P) = \emptyset \quad \frac{P \xrightarrow{(\nu\, \widetilde{n})\,\overline{m}\langle M\rangle} P_1' \quad P \xrightarrow{m(x)} P_2'}{!\, P \xrightarrow{\tau} ((\nu\, \widetilde{n})(P_1' \mid P_2'[x := M])) \mid !\, P} \ \widetilde{n} \cap \mathrm{fn}(Q) = \emptyset$$

**Table 2. Structural congruence, commitment and SOS clauses for late semantics for spi calculus.**

They are described in Table 3. The names **private** and **public** are reserved for the types of asymmetric keys, and in addition, the reserved name **void** is used as the parameter $m$ for operations which need no parameter, e.g., the **id** operation. There is a standard way of getting rid of such reserved names: Each time a reserved name is to be sent in the original encoding, first a number of fresh names are transmitted, namely exactly one for each reserved name in the original encoding. Then the actual communication takes place, using the fresh name that corresponds to the replaced reserved name. We stick to using reserved names since we feel that the essence of our encoding can be better presented in this way.

An agent (or term) operating on an encoded term typically generates a new name for the result channel, in order to avoid interference from concurrent accesses. For example, to check whether two terms located at $\ell$ and $m$ are equal, the construction $(\nu\, r)\, \overline{\ell}\langle \mathbf{match}, m, r\rangle.r().P$ could be used, which will continue as $P$ if the terms match. As a shorthand for $(\nu\, r)\, \overline{\ell}\langle c, m, r\rangle.P$ we often write $\overline{\ell}\langle c, m, \nu\, r\rangle.P$.

The type-orthodox reader may notice that our translation does not properly type the response channel $r$, which is sometimes nullary and sometimes unary. Types are out of the scope of this paper, but introducing a dummy name as object in the nullary case would remedy the situation.

In the presentation of the translation, we use $\pi$-calculus abstractions of the form $(\lambda\, x)\, P$ (ranged over by $F$) and applications, defined by $((\lambda\, x)\, P)(m) = P[x := m]$; we write $(\lambda\, m, n)\, P$ for $(\lambda\, m)\,(\lambda\, n)\, P$ and $F(m, n)$ for $F(m)(n)$. The translation also involves a case analysis on the terms occurring in the outermost operator of spi agents, and subterms occurring in terms. Each term (or subterm) which is not a variable must be recursively translated. (Variables are represented directly by names, which get substituted by term locations in inputs.) We write $M \Rightarrow F$ and $M, N \Rightarrow F$ for the translated spi calculus term(s) $M$ (and $N$) being used by the $\pi$-calculus abstraction $F$. This is formally defined as follows, where different cases are obtained depending on whether the parameter(s) of $F$ be variable(s) or not.

$$M \Rightarrow F = \begin{cases} (\nu\, m)([\![M]\!]_m \mid F(m)) \\ \qquad \text{if } M \text{ is not a variable} \\ F(M) \\ \qquad \text{if } M \text{ is a variable} \end{cases}$$

$$M, N \Rightarrow F = \begin{cases} (\nu\, m)\,(\nu\, n)([\![M]\!]_m \mid [\![N]\!]_n \mid F(m, n)) \\ \qquad \text{if } M \text{ and } N \text{ are not variables} \\ (\nu\, m)([\![M]\!]_m \mid F(m, N)) \\ \qquad \text{if } N \text{ is a variable but not } M \\ (\nu\, n)(F(M, n) \mid [\![N]\!]_n) \\ \qquad \text{if } M \text{ is a variable but not } N \\ F(M, N) \\ \qquad \text{if both are variables} \end{cases}$$

This notation will be used both in the following section which describes the translation of spi-calculus agents, and the the next which describes the translations of the terms occurring in agents.

### 3.1 Translating Agents

The parallel composition, replication, and restriction operators involve no terms, and the translation is homomor-

| Name | Term | Description |
|---|---|---|
| **match** | *all* | synchronize on $r$ if $m$ matches this term |
| **id** | names | return the name $n$ being encoded |
| **zero** | 0 | synchronize on $r$ if 0 is the encoded term |
| **pred** | $\mathrm{suc}(n)$ | return the location of $n$ |
| **text** | hashcodes | return the location of the text hashed |
| **fst** | pairs | return the location of the first component |
| **scd** | pairs | – or second component |
| **type** | asymmetric keys | return the type of key, **private** or **public** |
| **base** | asymmetric keys | return the location of $k$ in the key $k^+$ or $k^-$ |
| **k_match** | asymmetric keys | match $k$ against $k'$ for $k^+$ and $k'^-$ |
| **splain** | symmetric ciphertexts | return location of plaintext |
| **sk ey** | symmetric ciphertexts | return location of key |
| **pplain** | asymmetric ciphertexts | return location of plaintext |
| **pkey** | asymmetric ciphertexts | return location of key |
| **decrypt** | ciphertexts | if $m$ matches the encryption key, return the location of plaintext |

**Table 3. Reserved names for operations on translated terms**

phic: $[\![(\nu\, n)\, P]\!] = (\nu\, n)[\![P]\!]$, $[\![P \mid Q]\!] = [\![P]\!] \mid [\![Q]\!]$, and $[\![!\, P]\!] = ![\![P]\!]$.

Outputs and inputs use the original subject name as communication channel, but the communicated objects are *links* to encodings of terms.

$$
\begin{aligned}
[\![\overline{M}\langle N\rangle.P]\!] = M, N \Rightarrow (\lambda\, m, n)\, & \overline{m}\langle \mathbf{id}, \mathbf{void}, \nu\, r\rangle. && (1)\\
& r(i). && (2)\\
& \overline{i}\langle n\rangle. && (3)\\
& [\![P]\!] && (4)
\end{aligned}
$$

*Notes:* On line 1, ask the output subject for its identity; line 2, wait for a reply (carrying the identity, i.e., the name encoded); line 3, output the link to the output object, and on line 4, continue.

For the input prefix, note that the bound variable $x$ in the spi input is syntactically the same as the name $x$ in the $\pi$-calculus input.

$$
\begin{aligned}
[\![M\,(x).P]\!] = M \Rightarrow (\lambda\, m)\, & \overline{m}\langle \mathbf{id}, \mathbf{void}, \nu\, r\rangle.\\
& r(i).\\
& i(x).\\
& [\![P]\!]
\end{aligned}
$$

The remaining operators are dealt with using the appropriate operations on the terms involved. Note that for the integer **case** operator, the two parallel agents in the encoding are mutually exclusive: a term encoding never responds both on **zero** and **pred** operations. Again, the spi variables $x$ and $y$ bound in **let** and **case** are syntactically the same as the names $x$ and $y$ in the $\pi$-calculus translations.

$$
\begin{aligned}
[\![[M \textbf{ is } N]\, P]\!] = & \\
M, N \Rightarrow (\lambda\, m, n)\, & \overline{m}\langle \mathbf{match}, n, \nu\, r\rangle. && (1)\\
& r(). && (2)\\
& [\![P]\!]
\end{aligned}
$$

*Notes:* On line 1, tell the first term object to match against the second, and on line 2, wait for a reply (indicating that they did match).

$$
\begin{aligned}
[\![\textbf{let } (x, y) = M \textbf{ in } P]\!] = & \\
M \Rightarrow (\lambda\, m)\, & \overline{m}\langle \mathbf{fst}, \mathbf{v\,oid}, \nu\, r\rangle.r(x). && (1)\\
& \overline{m}\langle \mathbf{scd}, \mathbf{v\,oid}, \nu\, r\rangle.r(y). && (2)\\
& [\![P]\!]
\end{aligned}
$$

*Notes:* On line 1, ask for the first sub-object and wait for a link in return, and on line 2, do the same for the second sub-object.

$$
\begin{aligned}
[\![\textbf{case } M \textbf{ of } 0 : P\ \mathrm{suc}(x) : Q]\!] = & \\
M \Rightarrow (\lambda\, m)\, ( \quad & \overline{m}\langle \mathbf{zero}, \mathbf{v\,oid}, \nu\, r\rangle.r(). && (1)\\
& [\![P]\!]\\
& \mid\\
& \overline{m}\langle \mathbf{pred}, \mathbf{v\,oid}, \nu\, r\rangle.r(x).\\
& [\![Q]\!]\\
)&
\end{aligned}
$$

*Notes:* Ask the object if it is zero (line 1), and in parallel ask for a link to a predecessor object. Only one of the challenges will get a reply.

The decryption operator simply asks the encrypted term $M$ to decrypt using the key $N$, binding $x$ to the plaintext received:

$$
\begin{aligned}
[\![\textbf{case } M \textbf{ of } [x]_N \textbf{ in } P]\!] = & \\
M, N \Rightarrow (\lambda\, m, n)\, & \overline{m}\langle \mathbf{decrypt}, n, \nu\, r\rangle.\\
& r(x).\\
& [\![P]\!]
\end{aligned}
$$

### 3.2 Translating Terms

In spi calculus *terms* are the values sent and received. As mentioned earlier, their translation into $\pi$-calculus, $[\![M]\!]_\ell$, is

parameterised by $\ell$, which is a link for accessing the encoding of $M$. We continue to use the $M \Rightarrow F$ and $M, N \Rightarrow F$ notations to handle the case analysis on subterms.

Different types of terms handle different operations. If a term encoding is given an operation it does not handle, it will simply ignore the challenge and (possibly) let the "caller" deadlock. This corresponds to a type error in the source term/agent; again, typing is not in the scope of this paper. All term translations are replications, which handle concurrent accesses.

We start by giving the encodings of the two simplest terms: names $n$ and the constant $0$, which have no subterms.

$$
\begin{array}{lll}
[\![n]\!]_\ell = ! \, \ell(c, m, r). & & (1) \\
\quad \big( [c = \mathbf{id}] \, \overline{r}\langle n \rangle & & (2) \\
\quad + [c = \mathbf{match}] & & (3) \\
\quad\quad \overline{m}\langle \mathbf{id}, \mathbf{v\,oid}, \nu \, s \rangle. & & (4) \\
\quad\quad s(x). & & (5) \\
\quad\quad [x = n] \, \overline{r}\langle\rangle & & (6) \\
\quad \big) & &
\end{array}
$$

*Notes:* On line 1, replicate in order to handle concurrent requests; on line 2, handle an identity request by returning your true name; on line 3, handle a match request by asking the other end for its identity (line 4), await a reply (line 5), and if it's our identity then reply (line 6).

The encoding of zero is similar but simpler:

$$
\begin{array}{l}
[\![0]\!]_\ell = ! \, \ell(c, m, r). \\
\quad \big( [c = \mathbf{zero}] \, \overline{r}\langle\rangle \\
\quad + [c = \mathbf{match}] \\
\quad\quad \overline{m}\langle \mathbf{zero}, \mathbf{void}, r \rangle \\
\quad \big)
\end{array}
$$

The successor, hash code and asymmetric key terms have a single subterm, and use $M \Rightarrow F$ notation.

$$
\begin{array}{l}
[\![\mathrm{suc}(M)]\!]_\ell = \\
\quad M \Rightarrow (\lambda \, m) \, \mathbf{Unary}(\ell, \mathbf{pred}, m) \\
[\![\mathrm{hashc}(M)]\!]_\ell = \\
\quad M \Rightarrow (\lambda \, m) \, \mathbf{Unary}(\ell, \mathbf{text}, m) \\
[\![M^+]\!]_\ell = \\
\quad M \Rightarrow (\lambda \, m) \, \mathbf{PKey}(\ell, \mathbf{private}, m, \mathbf{public}) \\
[\![M^-]\!]_\ell = \\
\quad M \Rightarrow (\lambda \, m) \, \mathbf{PKey}(\ell, \mathbf{public}, m, \mathbf{private})
\end{array}
$$

The agent $\mathbf{Unary}$ handles the simplest unary terms, and is parameterised by the name of the subterm field; it handles requests for the subterm (line 1 below), and uses the agent $\mathbf{Unary\_Match}$ which handles $\mathbf{match}$ operations for unary terms. To handle $\mathbf{match}$, it asks the other object for its corresponding subobject (line 2 below) and tells the subobjects to match (line 3).

$$
\begin{array}{lll}
\mathbf{Unary}(\ell, sub, n) = & & \\
\quad ! \, \ell(c, m, r). & & \\
\quad \big( [c = sub] \, \overline{r}\langle n \rangle & & (1) \\
\quad + [c = \mathbf{match}] & & \\
\quad\quad \mathbf{Unary\_Match}(m, sub, n, r) & & \\
\quad \big) & & \\
\mathbf{Unary\_Match}(m, sub, n, r) = & & \\
\quad\quad \overline{m}\langle sub, \mathbf{v\,oid}, \nu \, s \rangle. & & (2) \\
\quad\quad s(x). & & \\
\quad\quad \overline{n}\langle \mathbf{match}, x, r \rangle & & (3)
\end{array}
$$

Asymmetric keys are encoded using $\mathbf{PKey}$, where the parameters $t$ and $u$ give the type of the key and its complementary type, and $n$ is the *key base*, i.e., the location of $M$ for a key $M^+$ or $M^-$.

$$
\begin{array}{lll}
\mathbf{PKey}(\ell, t, n, u) = & & \\
\quad ! \, \ell(c, m, r). & & \\
\quad \big( [c = \mathbf{type}] \, \overline{r}\langle t \rangle & & (1) \\
\quad + [c = \mathbf{base}] \, \overline{r}\langle n \rangle & & (2) \\
\quad + [c = \mathbf{match}] & & (3) \\
\quad\quad \overline{m}\langle \mathbf{type}, \mathbf{v\,oid}, \nu \, s \rangle. & & (4) \\
\quad\quad s(x). & & \\
\quad\quad [x = t] & & (5) \\
\quad\quad\quad \mathbf{Unary\_Match}(m, \mathbf{base}, n, r) & & \\
\quad + [c = \mathbf{k\_match}] & & (6) \\
\quad\quad \overline{m}\langle \mathbf{type}, \mathbf{v\,oid}, \nu \, s \rangle. & & \\
\quad\quad s(x). & & \\
\quad\quad [x = u] & & (7) \\
\quad\quad\quad \mathbf{Unary\_Match}(m, \mathbf{base}, n, r) & & \\
\quad \big) & &
\end{array}
$$

*Notes:* Lines 1 and 2 simply return the type and base of the key; on line 3 a match request is handled by asking the other object for its key type (line 4), checking that it's the same (line 5), and then matching the key bases. Line 6 handles a key match request similarly to a match, but now checking that the key types are complementary (line 7).

The remaining terms (pairs and encryptions) have two subterms, and thus use the $M, N \Rightarrow F$ notation.

$$
\begin{array}{lll}
[\![(M, N)]\!]_\ell = & & \\
M, N \Rightarrow (\lambda \, n_1, n_2) \, ! \, \ell(c, m, r). & & \\
\quad \big( [c = \mathbf{fst}] \, r\langle n_1 \rangle & & (1) \\
\quad + [c = \mathbf{scd}] \, r\langle n_2 \rangle & & (2) \\
\quad + [c = \mathbf{match}] & & (3) \\
\quad\quad \mathbf{Binary\_Match}( & & \\
\quad\quad\quad m, \mathbf{fst}, n_1, \mathbf{scd}, n_2, r & & \\
\quad\quad ) & & \\
\quad \big) & &
\end{array}
$$

*Notes:* Lines 1 and 2 handle requests for subcomponents, while line 3 handles match requests using the **Binary_Match** agent (see below).

$$[\![\{M\}_N]\!]_\ell = \\ M, N \Rightarrow (\lambda\, m, n) \\ \quad \mathbf{Cipher}(\ell, \mathbf{splain}, m, \mathbf{skey}, n, \mathbf{match})$$

$$[\![\{|M|\}_N]\!]_\ell = \\ M, N \Rightarrow (\lambda\, m, n) \\ \quad \mathbf{Cipher}(\ell, \mathbf{pplain}, m, \mathbf{pkey}, n, \mathbf{k\_match})$$

The **Binary_Match** agent is parameterised by the operations to access the subterms $sub_1$ and $sub_2$ of $m$, and simply "chains" a unary match for each.

$$\mathbf{Binary\_Match}(m, sub_1, n_1, sub_2, n_2, r) = \\ (\nu\, s)\,\big(\mathbf{Unary\_Match}(m, sub_1, n_1, s) \\ \quad \mid s().\mathbf{Unary\_Match}(m, sub_2, n_2, r) \\ \big)$$

Ciphertexts are created using symmetric or asymmetric ciphers, and use different operations to access plaintext and to match encryption keys.

$$\begin{array}{l} \mathbf{Cipher}(\ell, plaintext, p, key, k, k\_match) = \\ \quad !\,\ell(c, m, r). \\ \quad \big([c = plaintext]\, r\langle p\rangle \qquad (1) \\ \quad + [c = key]\, r\langle k\rangle \qquad (2) \\ \quad + [c = \mathbf{match}] \\ \quad\quad \mathbf{Binary\_Match}( \qquad (3) \\ \quad\quad\quad m, plaintext, p, key, k, r \\ \quad\quad ) \\ \quad + [c = \mathbf{decrypt}] \\ \quad\quad \overline{k}\langle k\_match, m, \nu\, s\rangle. \qquad (4) \\ \quad\quad s(). \\ \quad\quad r\langle p\rangle \qquad (5) \\ \big) \end{array}$$

*Notes:* Lines 1 and 2 handle requests for plaintext and key (i.e. **splain/pplain** and **skey/pkey**); line 3 handles match by matching both plaintext and key; line 4 handles decrypt requests by first key-matching the supplied key against our own (using **match** or **k_match** for symmetric or asymmetric keys, respectively), and returning the plaintext if they match (line 5).

### 3.2.1 Unique-Plaintext Ciphertexts

Under the assumption that each encryption of a fixed plaintext generates a new ciphertext, which is the case e.g. if random padding of the plaintext is used, the encoding can be optimized slightly, resulting in smaller state spaces: at line 1 below, a match is successful only if matching against the same term.

$$\begin{array}{l} \mathbf{Cipher}(\ell, plaintext, p, key\ , k, kmatch) = \\ \quad !\,\ell(c, m, r). \\ \quad \big([c = plaintext]\, r\langle p\rangle \\ \quad + [c = key]\, r\langle k\rangle \\ \quad + [c = \mathbf{match}] \\ \quad\quad [m = \ell]\, r\langle\rangle \qquad (1) \\ \quad + [c = \mathbf{decrypt}] \\ \quad\quad \overline{k}\langle k\_match, m, \nu\, s\rangle. \\ \quad\quad s(). \\ \quad\quad r\langle p\rangle \\ \big) \end{array}$$

## 4 Preservation of Spi Calculus May-Tests

Testing in the sense of De Nicola and Hennessy [13] rests on the idea of building an observation scenario for some description framework for concurrent processes by employing the expressive power of that framework itself. This goal is achieved by employing agents as so-called experiments, setting them up in concurrent interaction with agents that are to be tested and letting those experiments emit some signal if and when they have reached any success state. There is a substantial literature about testing over the $\pi$-calculus, for example by Boreale and De Nicola [12], and Abadi and Gordon identified may-testing already in [3] as particularly suitable for the spi calculus since it is generally associated with safety properties, the ones that are obviously most interesting in connection with spi. We prove that our translation is adequate with respect to may-**tests** in the sense that a spi calculus agent $P$ may pass a spi calculus experiment $E$ if and only if its translation, $[\![P]\!]$, may pass $E$'s translation, $[\![E]\!]$. We state next the formal definitions that are needed for our purposes and the adequacy theorem itself.

**Definition 1**

1. An *experiment* is an agent that may use a distinguished name \$. An action on \$ is a *success signal*.
2. A spi or $\pi$-calculus agent $P$ *may pass* an experiment $E$ if some sequence of $\tau$-steps of the composed agent $P \mid E$ has a state in which success is signalled. Formally, we denote this property by $P$ MAY $E$.

**Theorem 2** *Let $P$ and $E$ be a spi calculus agent and experiment, respectively. Then $P$ MAY $E$ if and only if $[\![P]\!]$ MAY $[\![E]\!]$.*

A lack of space prevents us from presenting the long and complex proof of this result. We refer the reader to [6] for a formal presentation, restricting ourselves here to a largely

informal overview. The result is essentially a consequence of several operational correspondence properties in whose proofs the actual work lies. By far the most difficult one of these properties is concerned with going from operational steps of any agent obtained by translation back to operational steps or commitments of its pre-image. The translation induces sequences of steps on the $\pi$-calculus side where there is only a commitment or a single step on the spi calculus side. Concurrent sequences of this kind can be interleaved and they can be composed of interleaved subsequences amongst which there is communication. Therefore we get a very complicated correspondence between the states of $[\![P]\!]$ and those of $P$, where $P$ is any spi agent. The decisive observation, however, is that it is not imperative to use a direct operational correspondence. Instead we use an indirect one, where we rearrange the states of $[\![P]\!]$ along the way so that they stay structurally more similar to those of $P$. This strategy requires that the rearrangement preserves bisimulation and two crucial properties of the translation are needed for that: First, it is compositional with respect to all static operators, that is, parallel composition, restriction and replication; second, the translations of terms can be viewed as *resources* in the sense of the Replication Lemmas over the $\pi$-calculus (see, for example, [19]), and we get suitable agent rearrangements by applying these lemmas.

For the core of the proof of Theorem 2, we need to use operational semantics without structural congruence since arbitrary rearrangements due to it would complicate the compositional arguments as they would have to be distinguished from rearrangements of the kind explained in the previous paragraph. In consequence, we use bisimulation over both the $\pi$- and the spi calculus to relate our rearrangements to actual states of spi agents and their translations. To this end, we carry over the notion of context bisimulation from higher-order process algebra [18] to both calculi. That gives us a uniform framework to work with and, moreover, it totally avoids the complications involved in other bisimilarities proposed for the spi calculus (see [10] for a good recent overview). Our immediate purposes are best served by the definitions below; the definitions that we actually use in the proof are equivalent, but are presented somewhat differently to make the proof go smoother.

**Definition 3** *Context bisimilarity* on spi calculus agents is defined to be the largest binary relation $\sim_{\mathrm{cxt}}$ so that $P \sim_{\mathrm{cxt}} Q$ implies:

**i.i.** Whenever $P \xrightarrow{(\nu\,\widetilde{n})\,\overline{m}\langle M\rangle} P'$, then $Q \xrightarrow{(\nu\,\widetilde{n})\,\overline{m}\langle M\rangle} Q'$ for some $Q'$ s.t. $(\nu\,\widetilde{n})(R\{x := M\} \mid P') \sim_{\mathrm{cxt}} (\nu\,\widetilde{n})(R\{x := M\} \mid Q')$ for every spi agent expression $R$ in which at most $x$ occurs as a free variable.

**i.ii.** Whenever $P \xrightarrow{m(x)} P'$, then $Q \xrightarrow{m(x)} Q'$ for some $Q'$ s.t. $P'\{x := M\} \sim_{\mathrm{cxt}} Q'\{x := M\}$ for every closed term $M$.

**i.iii.** Whenever $P \xrightarrow{\tau} P'$, then $Q \xrightarrow{\tau} Q'$ for some $Q'$ s.t. $P' \sim_{\mathrm{cxt}} Q'$.

**ii.i.-iii.** Like i.i-i.iii but with $Q$ driving the bisimulation game.

**Definition 4** *Context bisimilarity* on $\pi$-calculus agents is defined to be the largest binary relation $\sim_{\mathrm{cxt}}$ so that $P \sim_{\mathrm{cxt}} Q$ implies:

**i.i.** Whenever $P \xrightarrow{(\nu\,\widetilde{b'})\,\overline{a}\langle\widetilde{b}\rangle} P'$, then $Q \xrightarrow{(\nu\,\widetilde{b'})\,\overline{a}\langle\widetilde{b}\rangle} Q'$ for some $Q'$ s.t. $(\nu\,\widetilde{b'})(R\{\widetilde{c} := \widetilde{b}\} \mid P') \sim_{\mathrm{cxt}} (\nu\,\widetilde{b'})(R\{\widetilde{c} := \widetilde{b}\} \mid Q')$ for every $\pi$ agent $R$, assuming that the names in $\widetilde{c}$ are distinct with $|\widetilde{b}| = |\widetilde{c}|$.

**i.ii.** Whenever $P \xrightarrow{a(\widetilde{c})} P'$, then $Q \xrightarrow{a(\widetilde{c})} Q'$ for some $Q'$ s.t. $P'\{\widetilde{c} := \widetilde{b}\} \sim_{\mathrm{cxt}} Q'\{\widetilde{c} := \widetilde{b}\}$ for every $\widetilde{b}$ with $|\widetilde{c}| = |\widetilde{b}|$.

**i.iii.** Whenever $P \xrightarrow{\tau} P'$, then $Q \xrightarrow{\tau} Q'$ for some $Q'$ s.t. $P' \sim_{\mathrm{cxt}} Q'$.

**ii.i.-iii.** Like i.i-i.iii but with $Q$ driving the bisimulation game.

The operational correspondences together establish what could be regarded as a translation-coupled expansion between $P$ and $[\![P]\!]$ for any spi agent $P$. Conventional expansion, that is, expansion where the translation is not directly built into it, has played a major role in work on translating the asynchronous $\pi$- to the $\pi$I-calculus, the $\pi$-calculus where only private names are mobile [8]. But we need to formulate our expansion also on the basis of what we call the *ancestor relation* (cf. [6]). The ancestor relation seems to be absolutely necessary for us to handle concurrent threads of activity on the $\pi$-calculus side were each such thread corresponds to just a single step or commitment on the spi calculus side. We denote it by $\blacktriangleleft$ and some schematic clauses for it are collected in Table 4 on the following page, where $\Rightarrow$ stands for zero or more $\tau$-labelled transitions, $\stackrel{\mu}{\Rightarrow}$ for zero or more $\tau$-labelled transitions followed by a transition labelled with $\mu$. An essential property of the ancestor relation is that its defining clauses are compositional on the static operators, since that allows us to reason compositionally. The other essential aspect is conveyed by those clauses that are of the form

$$\frac{[\![P]\!] \Rightarrow Q >_{\varphi} \stackrel{\mu}{\Rightarrow} \sim_{\mathrm{cxt}} [\![P']\!]}{P \blacktriangleleft Q},$$

where the outermost operator of $P$ is a match, $\mathbf{let}$, $\mathbf{case}$ or prefix. They are to be read as follows: It holds that $P \blacktriangleleft Q$ if

**i.** $Q$ is equal to $[\![P]\!]$ or an intermediate state in the $\pi$-calculus execution trace of $P$'s outermost operator and

$$\frac{[\![[M \text{ is } N]\, P]\!] \Rightarrow Q >_{M=N} \stackrel{\tau}{\Rightarrow}\sim_{\mathrm{cxt}} [\![P]\!]}{[M \text{ is } N]\, P \blacktriangleleft Q} \quad (\blacktriangleleft\text{-MATCH})$$

$$\frac{[\![\mathbf{let}\ (x_1, x_2) = M\ \mathbf{in}\ P]\!] \Rightarrow Q >_{M=(M_1,M_2)} \stackrel{\tau}{\Rightarrow}\sim_{\mathrm{cxt}} [\![P[(x_1, x_2) := (M_1, M_2)]]\!]}{\mathbf{let}\ (x_1, x_2) = M\ \mathbf{in}\ P \blacktriangleleft Q} \quad (\blacktriangleleft\text{-PAIR})$$

$$\frac{[\![\mathbf{case}\ 0\ \mathbf{of}\ 0 : P_1\ \mathrm{suc}(x) : P_2]\!] \Rightarrow Q >_{\mathrm{true}} \stackrel{\tau}{\Rightarrow}\sim_{\mathrm{cxt}} [\![P_1]\!]}{\mathbf{case}\ 0\ \mathbf{of}\ 0 : P_1\ \mathrm{suc}(x) : P_2 \blacktriangleleft Q} \quad (\blacktriangleleft\text{-ZERO})$$

$$\frac{[\![\mathbf{case}\ M\ \mathbf{of}\ 0 : P_2\ \mathrm{suc}(x) : P_2]\!] \Rightarrow Q >_{M=\mathrm{suc}(M_1)} \stackrel{\tau}{\Rightarrow}\sim_{\mathrm{cxt}} [\![P_2[x := M_1]]\!]}{\mathbf{case}\ M\ \mathbf{of}\ 0 : P_1\ \mathrm{suc}(x) : P_2 \blacktriangleleft Q} \quad (\blacktriangleleft\text{-SUC})$$

$$\frac{[\![\mathbf{case}\ M\ \mathbf{of}\ [x]_K\ \mathbf{in}\ P]\!] \Rightarrow Q >_{\varphi} \stackrel{\tau}{\Rightarrow}\sim_{\mathrm{cxt}} [\![P[x := M_1]]\!]}{\mathbf{case}\ M\ \mathbf{of}\ [x]_K\ \mathbf{in}\ P \blacktriangleleft Q} \quad (\blacktriangleleft\text{-DEC})$$

$\varphi$ in the above rule stating that (a) $M = \{M_1\}_K$ or (b) $M = \{|M_1|\}_{N^p}$ and $K = (N^p)^{-1}$ for some $N, p = -, +$

$$\mathbf{0} \blacktriangleleft \mathbf{0} \quad (\blacktriangleleft\text{-NIL})$$

$$\frac{P_1 \blacktriangleleft Q_1 \quad P_2 \blacktriangleleft Q_2}{P_1 \mid P_2 \blacktriangleleft Q_1 \mid Q_2} \quad (\blacktriangleleft\text{-PAR})$$

$$\frac{P \blacktriangleleft Q}{(\nu\, m)\, P \blacktriangleleft (\nu\, m)\, Q} \quad (\blacktriangleleft\text{-NEW})$$

$$!\, P \blacktriangleleft ![\![P]\!] \quad (\blacktriangleleft\text{-REP})$$

**Table 4. Selected schematic clauses for the ancestor relation on spi and $\pi$-terms.**

**ii.** if $\varphi$ holds, then $Q \stackrel{\mu}{\Rightarrow}\sim_{\mathrm{cxt}} [\![P']\!]$.

The intuition is that $P \blacktriangleleft Q$ holds if $Q$ is bisimilar to a successor state of $[\![P]\!]$ in which no thread of activity that corresponds to an unguarded occurrence of a match, **let**, **case** or prefix construct in $P$ has been completed.

The central and most difficult part of the proof of Theorem 2 then consists of establishing a "backward" operational correspondence that goes from steps of $[\![P]\!]$ to steps and commitments of $P$, where $P$ is any spi agent. This operational correspondence is coupled via the ancestor relation and involves context bisimilarity as for term rearrangements. There is also a "forward" operational correspondence that goes from steps $P$ to steps of $[\![P]\!]$, and there are auxiliary operational correspondences that bridge the gap between operational semantics with and without structural congruence.

As a final remark, we note that, while context bisimilarity is new to the spi calculus, it has recently found proof technical application also to Cardelli and Gordon's mobile ambients [15].

## 5 Related and Future Work

Our translation from the spi to the $\pi$-calculus is obviously related to the general question of how to express encryption and other security concepts directly in the $\pi$-calculus, foregoing the spi calculus or any other higher-level framework. In the appendix to [3] Abadi and Gordon discuss three different approaches, where these schemes might partly be considered embryonic versions of what we get via our translation on the $\pi$-calculus side. In particular the technique of representing a piece of data by an agent that is to be accessed via a dedicated link, which goes back to the early $\pi$-calculus literature, is already proposed as useful in [4]. There is, however, no explicit translation from the spi to the $\pi$-calculus in [4] and therefore of course also no adequacy result along the lines of Theorem 2. Moreover, only a very limited range of features is discussed with respect to their expressibility in the $\pi$-calculus whereas we effectively deal with everything that belongs to the spi calculus as it was originally presented. It can nevertheless already be concluded from [4] that in the way we are using the $\pi$-calculus as a kind of assembler language for implementing the spi calculus, we can probably never hope to obtain full abstraction: There will probably always be $\pi$-calculus experiments that distinguish the respective translations of any two agents even if they are testing equivalent with respect to spi calculus experiments. That is in fact easily confirmed as far as our concrete translation is concerned: Let $P = (\nu\, x)\, \overline{a}\langle\{b\}_x\rangle.\mathbf{0}$ and $Q = (\nu\, y)\, \overline{a}\langle\{c\}_y\rangle.\mathbf{0}$. Since neither $x$ nor $y$ are revealed, no spi-experiment will be able to decipher $b$ or $c$ after it has received $\{b\}_x$ or $\{c\}_y$, respectively, so $P$ and $Q$ are may-equivalent (as spi agents). But $[\![P]\!]$ and $[\![Q]\!]$ are not may-equivalent (as $\pi$-agents). A $\pi$-experiment can distinguish them by accessing plaintexts directly via the **splain**-method, and then test that what is returned for equality either to $b$ or $c$, e.g., $R = a(e)\,.\,\overline{e}\langle\mathbf{splain}, \mathbf{void}, \nu\, r\rangle\,.\,r(x)\,.\,[x = b]\,\$$.

Another idea discussed in [4], and also by Carbone and Maffeis in [11], is to model the communication of encrypted data on the basis of extending the $\pi$-calculus by multi-name synchronisation, that is, a channel may consist of what may perhaps be seen as several sub-channels that all take part in any synchronisation on the channel at once. This approach

avoids at least to some degree the above-mentioned problem with stronger forms of adequacy than the preservation of may- and must-testing. The downside is of course that the target framework is less fundamental than the original $\pi$-calculus and also that there do not seem to be any automated $\pi$-calculus verification tools that support multi-name synchronisation in the input.

We have implemented our translation in a prototype that generates input for the Mobility Workbench [20]. We note that a translated spi calculus test $[\![P \mid E]\!]$ will have finitely many states reachable by $\tau$-moves if $P \mid E$ already already has the same property. The reason is (a) that only those parts of $[\![P \mid E]\!]$ that correspond to terms are theoretically problematic as they may harbour the only occurrences of replications not already present on the spi side where, however, (b) any thread of activity within them is triggered by activity within some agent part and then guaranteed to terminate. Our prototype can therefore be used to translate both an agent and an experiment, whereupon in many practical cases the Mobility Workbench can check whether the agent may pass the experiment or not. As for work related to that, we are aware of work based on [10] that aims at tools for symbolic bisimulation for the spi calculus. Also, the spi calculus has been considered for an extension of a logic programming implementation of the $\pi$-calculus in [17].

As for possible future work, it would be very interesting to adapt the encoding to other security-related calculi such as the applied $\pi$-calculus [1]. We also want to carry our further experiments and case studies with our prototype, and enhance it so that we can make it publicly available. Still another issue is to extend the prototype so as to produce output for backends other than the Mobility Workbench, such as for example for the MIHDA toolkit [14].

**Acknowledgement** We would like to thank Martín Abadi, Andrew Gordon and Emilio Tuosto for discussions about earlier versions of this paper. We would also like to thank the anonymous referees for their remarks and suggestions.

## References

[1] M. Abadi and C. Fournet. Mobile Values, New Names, and Secure Communication. In *Principles of Programming Languages*, pages 104–115. ACM, 2001. POPL '01 symposium proceedings.

[2] M. Abadi and A. Gordon. The Spi Calculus. In *Computer and Communications Security*, pages 36–47. ACM, 1997. Conference proceedings.

[3] M. Abadi and A. Gordon. A Calculus for Cryptographic Protocols: The Spi Calculus. Technical Report 149, SRC, Palo Alto, California, 1998.

[4] M. Abadi and A. Gordon. A Calculus for Cryptographic Protocols: The Spi Calculus. *Information and Computation*, 148(1):1–70, 1999.

[5] R. Amadio, D. Lugiez, and V. Vanackère. On the Symbolic Reduction of Processes with Cryptographic Functions. *Theoretical Compututer Science*, 290(1):695–740, 2003.

[6] M. Baldamus, J. Parrow, and B. Victor. Spi Calculus Translated to $\pi$-Calculus Preserving May-Testing. Technical Report 2003-063, Department of Information Technology, Uppsala University, Sweden, 2003.

[7] B. Blanchet. An Efficient Cryptographic Protocol Verifier based on Prolog Rules. In *Computer Security Foundations*, pages 82–96. IEEE, 2001. CSFW-14 proceedings.

[8] M. Boreale. On the Expressiveness of Internal Mobility in Name-Passing Calculi. *Theoretical Computer Science*, 195:205–226, 1998.

[9] M. Boreale and M. Buscemi. Experimenting with STA, a Tool for Automatic Analysis of Security Protocols. In *Applied Computing*, pages 281–285. ACM, 2002. SAC '02 proceedings.

[10] J. Borgström and U. Nestmann. On Bisimulations for the Spi Calculus. Technical Report IC/2003/34, EPFL I&C, Lausanne, Switzerland, 2003.

[11] M. Carbone and S. Maffeis. On the Expressive Power of Polyadic Synchronisation in $\pi$-Calculus. *Nordic Journal of Computing*, 10(2):70–98, 2003.

[12] R. De Nicola and M. Boreale. Testing Equivalences for Mobile Processes. *Information and Computation*, 120:279–303, 1995.

[13] R. De Nicola and M. Hennessy. Testing Equivalences for Processes. *Theoretical Computer Science*, 34:83–133, 1983.

[14] G. Ferrari, U. Montanari, R. Raggi, and E. Tuosto. From Co-Algebraic Specifications to Implementation: The MIHDA Toolkit. In *Formal Methods for Components and Objects*, LNCS. Springer-Verlag, 2003. FMCO '03 symposium proceedings.

[15] M. Merro and F. Zappa Nardelli. Bisimulation Proof Methods for Mobile Ambients. In *Automata, Logic and Programming*, LNCS 2719, pages 584–598. Springer-Verlag, 2003. ICALP '03 proceedings.

[16] R. Milner, J. Parrow, and D. Walker. A Calculus of Mobile Processes, Parts I+II. *Information and Computation*, 100:1–77, 1992.

[17] Ping Yang, C. Ramakrishnan, and S. Smolka. A Logical Encoding of the $\pi$-Calculus: Model Checking Mobile Processes Using Tabled Resolution. Available via http://www.cs.sunysb.edu/∼lmc/mmc, 2003.

[18] D. Sangiorgi. Bisimulation in Higher-Order Calculi. *Information and Computation*, 131:141–178, 1996.

[19] D. Sangiorgi and D. Walker. *The $\pi$-Calculus: A Theory of Mobile Processes*. Cambridge University Press, 2003.

[20] B. Victor and F. Moller. The Mobility Workbench – A Tool for the $\pi$-Calculus. In *Computer Aided Verification*, LNCS 818, pages 428–440. Springer-Verlag, 1994. CAV '94 proceedings.