

# Authentication and Sandboxing in a Distributed $\pi$ -Calculus

Hans Hüttel\* and Morten Kühnrich\*\*

Department of Computer Science, Aalborg University  
Frederik Bajersvej 7E, 9220 Aalborg Ø, Denmark

**Abstract.** This paper presents an extension of Hennessy et al.  $D\pi$ -calculus [10] with constructs for signing and authenticating code and for sandboxing. A sort system, built on Milner's [8] sort systems for the polyadic  $\pi$ -calculus, is presented and proven sound with respect to an error predicate which ensures that errors do not occur outside sandboxes and that authentication and migration only happen when allowed. Furthermore a weak subject reduction result involving partial well sort-edness is presented.

## 1 Introduction

In computer networks the notion of sharing resources between sites has become increasingly common. This now also includes mobile code such as Java applets and downloadable applications for handheld devices. However, applications of unknown origin pose a potential security threat, as seen by the well-known examples of problems caused by malicious software.

Sandboxing is a known way to handle untrusted applications. In the Java Virtual Machine (JVM) [6] only trusted code has full permissions to different vital system resources such as the file system. All other untrusted code is sandboxed. In JVM 1.1 *digital signatures* were introduced as a means of authentication. Signed remote applets are only granted execution rights if the signature is trusted, i.e., the public key used to verify the signature must be trusted. Unsigned applets are sandboxed right away. Beginning with JVM 1.1 security policies regulating the use of system resources for trusted code (both local code and well signed remote applets) can be defined. These permissions are defined by the user or an administrator. Each permission gives a right to use one resource such as read or write permission to certain files and directories or host and ports.

Within the universe of process calculi various theories have been developed in order to model different security aspects of concurrent systems. In particular, there has been a substantial amount of work on process calculi with sites and mobility [1, 10, 12, 4]. Prominent examples are Mobile Ambients [1] and  $D\pi$ [5].

There has already been work on expressing signed code directly in process calculi.

---

\* hans@cs.aau.dk

\*\* mokyhn@cs.aau.dk

Merro and Hennessy present an extension of the ambient calculus with passwords in [7]. Here, the primitives of Mobile Ambients have been augmented with passwords so that mobility between sites is conditional on the presence of passwords.

M. Bugliesi et al. [2] introduce an ambient variant in which ambients may be signed. Under the use of a type system they obtain a secrecy result informally stating that if a piece of data is secret it cannot be revealed to the public.

Moreover, a number of authors have address the issue of sandboxing in a process calculus setting.

In [13] Vivas introduces a blocking operator in a  $\pi$ -calculus without locations and shows how one may use it to describe restrictions on access rights.

P. Sewell and J. Vitek show how interference of communication can be controlled using a boxed  $\pi$ -calculus in [12]. Their semantics allows the construction of so called wrappers, closely related to the concept of sandboxes.

In this paper we present an extension of the  $D\pi$ -calculus called  $DS\pi$  (the letter  $s$  can stand for secure, signed or sandboxed). The  $DS\pi$ -calculus, introduced in Section 2, allows the signing of processes under keys as well as sandboxing.

In Section 3 we present a type system in the form of a sort system extended to handle migration and authentication. A well-sorted process in our sort system will not error due to communication outside sandboxes *or* due to illegal migration or authentication. Processes can be *well-sorted* or *partially well-sorted*. The sort system satisfies a number of desirable properties. Firstly, we establish two subject reduction results. The standard subject reduction result states that well-sorted processes stay well-sorted under reductions, and a new, weak subject reduction result states that partially well-sorted processes can become well-sorted. Secondly, we define an error predicate and show that a well-sorted process will never misbehave.

## 2 Syntax and semantics of the $DS\pi$ -calculus

### 2.1 Syntax

The syntax in Table 1 is largely an extension of that of Hennessy and Riely's  $D\pi$  [10] with a few modifications. Our syntax uses simple tuples where  $D\pi$  uses nested tuples in message passing. Moreover, the biconditional match construct used in [10] has been replaced by *match* and *mismatch* since structural congruence fails to be type-safe in the presence of the biconditional axiom  $[\tilde{u} = \tilde{u}]P, Q \equiv P$  for arbitrary  $Q$ . In  $DS\pi$ , names represent channels, locations or keys (signatures). We let the letter  $\mathcal{N}$  denote a countable set of names  $a, b, c, \dots, m$ .

The letter  $\mathcal{V}$  denotes a set of variables  $x, y, z, \dots$  which is disjoint from  $\mathcal{N}$ . Identifiers  $u, v$  range over the set  $\mathcal{I} \stackrel{\text{def}}{=} \mathcal{N} \cup \mathcal{V}$ . A tilde like in  $\tilde{v}$  denotes a finite tuple of identifiers.

The syntax of  $DS\pi$  consists of *processes* and *networks*. Networks are built from located processes which we refer to as *sites* or *locations*. The location

$\gamma$	$::= \circ \mid \bullet$	Open and closed sites
$\pi$	$::= u?(\tilde{x} : T)$	Input
	$\mid u!\langle\tilde{v}\rangle$	Output
	$\mid \mathbf{go} \ v_\gamma$	Migration to location $v$
	$\mid \{P\}_v$	Encryption of $P$ under key $v$
	$\mid \mathbf{auth}_K(v_1, v_2)$	Authentication
$P$	$::= 0$	Inaction
	$\mid P \mid Q$	Parallel composition
	$\mid (\nu a : T) P$	Name restriction
	$\mid \pi.P$	Prefixed process
	$\mid *P$	Replication
	$\mid [\tilde{u} = \tilde{v}]P$	Match
	$\mid [\tilde{u} \neq \tilde{v}]P$	Mismatch
$M, N$	$::= 0$	Nil network
	$\mid M \mid N$	Parallel composition
	$\mid (\nu a : T)N$	Name restriction
	$\mid l[P]_\gamma$	Sites

**Table 1.** Syntax of the DS $\pi$ -calculus

structure of the calculus is flat, i.e. sites do not contain subsites. Communication is purely local to sites. Sites are tagged and can be either *open* (the tag  $\circ$ ) or *closed* (the tag  $\bullet$ ). An open site semantically allows all kinds of communication and migration whereas closed sites (sandboxes) do not allow processes to leave. A signed process with an unknown key at the moment of authentication can only end up at a closed site.

Messages are received using the prefix  $u?(\tilde{x} : T).P$  along channel  $u$ . Sometimes the sort annotation is left out and we write  $u?(\tilde{x}).P$ . A message received is instantiated for the variables in  $\tilde{x}$  in  $P$ . We demand that  $\tilde{x}$  is linear, i.e. variables in  $\tilde{x}$  occur at most once, and require that the names in  $\tilde{x}$  have the object sort of  $T$  (defined in Section 3). An output prefix  $u!\langle\tilde{v}\rangle.P$  transmits message  $\tilde{v}$  on the channel  $u$  and proceeds as  $P$ .

A name  $m$  within a process  $P$  can be restricted by  $(\nu m : T)P$  where the name  $m$  is now bound in  $P$  and required to have sort  $T$ .

Migration  $\mathbf{go} \ v_\gamma.P$  moves process  $P$  to the open location  $v$  if  $\gamma$  equals  $\circ$  and otherwise  $P$  is moved to the closed location  $v$ .

The new features of DS $\pi$  are the primitives for signing and authentication. The prefix  $\{P\}_v$  denotes that process  $P$  is signed with the key  $v$ . The authentication prefix is  $\mathbf{auth}_K(v_1, v_2)$ , where the set of recognized keys  $K \subseteq \mathcal{I}$  is assumed to be finite. Authentication is possible whenever a term  $\{P\}_k$  appears in parallel with  $\mathbf{auth}_K(v_1, v_2)$ . If the key  $k$  is found in the set of keys  $K$  then  $P$  is moved to the open location  $v_1$ , otherwise  $P$  is moved to the closed location  $v_2$ .

Restriction at network level is denoted by  $(\nu_l m : T)M$ , where the name  $m$  is restricted at location  $l$  in the network  $M$  to the sort  $T$ .

Restriction and input are the binding constructs of  $\text{DS}\pi$ . The notations  $\text{bn}(P)$  and  $\text{bn}(M)$  denote the set of bound identifiers for the process  $P$  or the network  $M$  respectively. For processes, these sets are defined in the standard way [11] with the added clause for the new encryption prefix that  $\text{bn}(\{P\}_v) = \text{bn}(P) \cup \{v\}$  and  $\text{fn}(\text{auth}_K(v_1, v_2)) = \{v_1, v_2\} \cup \text{fn}(K)$ . Further, we shall assume that the scope of names bound in  $P$  in the prefix  $\{P\}_v$  is  $P$ . The set of all identifiers is defined as  $\text{n}(P)$  or  $\text{n}(M)$  for processes and network respectively.

## 2.2 Semantics

**Substitution and structural congruence.** The substitution of a free name  $m$  for a free variable  $v$  written  $[m/v]$  is defined as expected. For signed processes we have that  $\{P\}_u.Q[m/v] = \{P[m/v]\}_u.Q[m/v]$ . Substitution is extended in a natural way to handle simultaneous substitutions of the form  $[\tilde{v}/\tilde{x}]$ .

We write  $P \equiv_\alpha Q$  or  $M \equiv_\alpha N$  if processes  $P$  and  $Q$  are equal up to renaming of bound names or networks  $M$  and  $N$  are equal up to renaming of bound names respectively.

**Definition 1.** *The network contexts are defined by*

$$\mathcal{E} ::= (-) \mid (\nu_l a : T) \mathcal{E} \mid \mathcal{E} \mid M$$

*The notation  $\mathcal{E}(N)$  means that  $N$  is inserted in the hole  $(-)$  in  $\mathcal{E}$ .*

The relation of structural congruence on networks  $M$  and  $N$  is the least binary reflexive, symmetric and transitive relation satisfying the rules given in Table 2.

**A reduction semantics for  $\text{DS}\pi$ .** The reduction semantics in Table 3 is an extended version of the semantics of  $\text{D}\pi$  given in [10]. Rule (R-GO) describes migration from an open site (hence a closed site does not permit escape by migration) and rule (R-COM) describes communication local to a site. The rules (R-AUTH1) and (R-AUTH2) describe authentication from open sites (hence a closed site does not permit authentication of signed processes). If  $k$  is in the set of known keys  $K$  then  $P$  will be placed at the open location  $l_1$ . If  $k$  is unknown (not in  $K$ ) then  $P$  is executed in a sandbox  $l_2$ . The remaining rules are standard.

**A labelled semantics for  $\text{DS}\pi$ .** In our results about the sort system we refer to the keys known in a system (Definition 7). To capture this, we introduce a labelled semantics for processes and networks.

We define the following set of action labels. An action is either a prefix  $\pi$  (which corresponds to all the possible actions in processes) or a located action  $\pi@l$  (which are all actions  $\pi$  at some location  $l$  in the network). We let  $\alpha$  denote the action labels:

$$\alpha ::= \pi \mid \pi@l$$

The labelled semantics of  $\text{DS}\pi$  is given in Table 4.

---

(S1) $N \mid 0 \equiv N$ (S3) $M \mid N \equiv N \mid M$ (S5) $l[P \mid Q]_\gamma \equiv l[P]_\gamma \mid l[Q]_\gamma$ (S7) $(\nu_l n : T) 0 \equiv 0$ (S9) $l[[\tilde{u} = \tilde{u}] P]_\gamma \equiv l[P]_\gamma$ (S11) $\frac{M \equiv N \quad N \equiv N'}{M \equiv N'}$ (S14) $M \mid (\nu_l n : T) N \equiv (\nu_l n : T) (M \mid N), \quad n \notin \text{fn}(M)$	(S2) $(M \mid N) \mid N' \equiv M \mid (N \mid N')$ (S4) $l[0]_\gamma \equiv 0$ (S6) $l[*P]_\gamma \equiv l[P \mid *P]_\gamma$ (S8) $l[(\nu n : T) P]_\gamma \equiv (\nu_l n : T) l[P]_\gamma, \quad n \neq l$ (S10) $l[[\tilde{u} \neq \tilde{v}] P]_\gamma \equiv l[P]_\gamma, \quad \text{when } \tilde{u} \neq \tilde{v}$ (S12) $\frac{M \equiv_\alpha N}{M \equiv N}$ (S13) $\frac{M \equiv N}{\mathcal{E}(N) \equiv \mathcal{E}(M)}$
---	---

---

**Table 2.** Structural congruence relation defined on networks

**Theorem 1 (Operational correspondence).** *The reduction semantics and the labelled transition semantics agree; that is*

$$M \longrightarrow M' \text{ iff } M \xrightarrow{\tau} M'$$

The action  $u?(\tilde{x} : T)$  is the input transition, a message may be received at the channel  $u$ , and instantiated for the variable  $\tilde{x}$ . The action  $u!\langle\tilde{v}\rangle$  is the corresponding output of  $\tilde{v}$  on channel  $u$ . The prefix  $\text{go } v_\gamma$  is the migration action where process  $P$  migrates to open or closed (depending on  $\gamma$ ) location  $v$ . The  $\text{auth}_K(v_1, v_2)$  and  $\{P\}_v$  are actions which signify authentication and the signing of a process. The notation  $\text{n}(\alpha)$  returns all names  $n \in \mathcal{N}$  occurring in  $\alpha$ .

There are two rules for sites. Rule (L-SITE1) allows any actions  $\pi$  at open locations whereas rule (L-SITE2) prohibits migration from closed locations.

*Example 1.* (A program server example in  $\text{DS}\pi$ ) Define a network consisting of a program server **ProgServer** and a system process **System**. The system sends a request for a signed process to the server. The program server returns a signed process which is authenticated in the system.

$$\begin{array}{l} \text{ProgServer} \quad [req?(x).\text{go } x_\circ.\{P\}_k]_\circ \mid \\ \text{System} \quad [\text{go ProgServer}_\circ.req!\langle\text{System}\rangle \mid \text{auth}_{\{k\}}(l, l_{box})]_\circ \end{array}$$

The messenger with the request  $req!\langle\text{System}\rangle$  migrates to the server

$$\begin{array}{l} \text{ProgServer} \quad [req?(x).\text{go } x_\circ.\{P\}_k]_\circ \mid \text{ProgServer}[req!\langle\text{System}\rangle]_\circ \mid \\ \text{System} \quad [\text{auth}_{\{k\}}(l, l_{box})]_\circ \end{array}$$

---

(R-GO)	$l[\mathbf{go} \ m_\gamma.P]_\circ \rightarrow m[P]_\gamma$
(R-COM)	$l[a!\langle\tilde{v}\rangle.P]_\gamma \mid l[a?(x:T).Q]_\gamma \rightarrow l[P]_\gamma \mid l[Q[\tilde{v}/x]]_\gamma$
(R-AUTH1)	$l[\mathbf{auth}_K(l_1, l_2).P]_\circ \mid l[\{P'\}_k.Q]_\circ \rightarrow l[P]_\circ \mid l[Q]_\circ \mid l_1[P']_\circ$ , if $k \in K$
(R-AUTH2)	$l[\mathbf{auth}_K(l_1, l_2).P]_\circ \mid l[\{P'\}_k.Q]_\circ \rightarrow l[P]_\circ \mid l[Q]_\circ \mid l_2[P']_\bullet$ , if $k \notin K$
(R-PAR)	If $N \rightarrow N'$ then $M \mid N \rightarrow M \mid N'$
(R-RES)	If $N \rightarrow N'$ then $(\nu_l \ n : T) \ N \rightarrow (\nu_l \ n : T) \ N'$
(R-STRUCT)	If $N \equiv M, M \rightarrow M'$ and $M' \equiv N'$ then $N \rightarrow N'$

---

**Table 3.** Reduction semantics of DS $\pi$  defined on networks

Next the source address **System** is revealed to the program server **ProgServer**:

$$\begin{array}{l} \mathbf{ProgServer} \ [\mathbf{go} \ \mathbf{System}_\circ.\{P\}_k]_\circ \mid \mathbf{ProgServer}[\ ]_\circ \mid \\ \mathbf{System} \ [\mathbf{auth}_{\{k\}}(l, l_{box})]_\circ \end{array}$$

The new signed application  $\{P\}_k$  is returned to **System**:

$$\mathbf{System}[\mathbf{auth}_{\{k\}}(l, l_{box})]_\circ \mid \mathbf{System}[\{P\}_k]_\circ$$

Since  $k$  is a known key this evaluates to

$$\mathbf{System}[\ ]_\circ \mid \mathbf{1}[P]_\circ$$

*Example 2 (Control of system resources).* Below a simple file system is modelled. A file with filename  $n$  and content  $c$  is created using **CreateFile**

$$\begin{array}{l} \mathbf{CreateFile}(n, c) \stackrel{\text{def}}{=} \\ *req?(s, fname).[fname = n]\mathbf{go} \ s_\circ.io!\langle c \rangle \end{array}$$

In order to look up a file, we send a receiving address and a file name on the channel  $req$ . If there is a file with that name the expression  $\mathbf{go} \ s_\circ.io!\langle c \rangle$  is triggered and the file (in the example represented by a name) is delivered on the channel  $io$  at the location  $s$ . Below a file system which has the capabilities of reading and writing files is defined.

$$\begin{array}{l} \mathbf{IOSystem} \stackrel{\text{def}}{=} \\ \mathbf{Files}[\ ]_\circ \mid \\ \mathbf{ReadFile}[*req?(from, fname).\mathbf{go} \ \mathbf{Files}_\circ.io!\langle from, fname \rangle] \mid \\ \mathbf{WriteFile}[*req?(n, c).\mathbf{go} \ \mathbf{Files}_\circ.\mathbf{CreateFile}(n, c)] \end{array}$$

---


$$\begin{array}{l}
\text{(L-PRE)} \frac{}{\pi.P \xrightarrow{\pi} P} \quad \text{(L-STRUCT)} \frac{N \equiv M \quad M \xrightarrow{\alpha} M' \quad M' \equiv N'}{N \xrightarrow{\alpha} N'} \\
\text{(L-COM)} \frac{P \xrightarrow{m(\tilde{v})} P' \quad Q \xrightarrow{m(\tilde{x}:T)} Q'}{l[P]_{\gamma} \mid l[Q]_{\gamma} \xrightarrow{\tau} l[P']_{\gamma} \mid l[Q']_{\gamma}[\tilde{v}/\tilde{x}]} \\
\text{(L-AUTH1)} \frac{M \xrightarrow{\text{auth}_K(l_1, l_2)@l} M' \quad N \xrightarrow{\{P'\}_k @l} N'}{M \mid N \xrightarrow{\tau} M' \mid N' \mid l_1[P']_{\circ}}, k \in K \\
\text{(L-AUTH2)} \frac{M \xrightarrow{\text{auth}_K(l_1, l_2)@l} M' \quad N \xrightarrow{\{P'\}_k @l} N'}{M \mid N \xrightarrow{\tau} M' \mid N' \mid l_2[P']_{\bullet}}, k \notin K \\
\text{(L-GO)} \frac{P \xrightarrow{\text{go } m\gamma} P'}{l[P]_{\circ} \xrightarrow{\tau} m[P']_{\gamma}} \quad \text{(L-PAR)} \frac{N \xrightarrow{\alpha} N'}{M \mid N \xrightarrow{\alpha} M \mid N'} \\
\text{(L-RES)} \frac{N \xrightarrow{\alpha} N'}{(\nu n : T)N \xrightarrow{\alpha} (\nu n : T)N'}, n \notin n(\alpha) \\
\text{(L-SITE1)} \frac{P \xrightarrow{\alpha} P'}{l[P]_{\circ} \xrightarrow{\alpha @l} l[P']_{\circ}} \\
\text{(L-SITE2)} \frac{P \xrightarrow{\alpha} P'}{l[P]_{\bullet} \xrightarrow{\alpha @l} l[P']_{\bullet}}, \alpha \in \{m?(\tilde{x}), m!(\tilde{v})\}
\end{array}$$


---

**Table 4.** Labelled transition semantics for DS $\pi$  processes and networks

In order to write a file a messenger migrates to the collection of all files at the site `Files` and perform a writing request  $req\langle n, c \rangle$ ; write file with contents  $c$  under the name  $n$ . The following network  $M$  creates a file with the content  $a$  under the file name  $myFile$

$$M \stackrel{\text{def}}{=} IOSystem \mid l[\text{go WriteFile}_{\circ}.req\langle myFile, a \rangle].$$

**Definition 2.** The weak labelled transition relation  $M \xrightarrow{\alpha} M'$  is defined by

$$M \xrightarrow{\tau}^* N \xrightarrow{\alpha} N' \xrightarrow{\tau}^* M'$$

where  $\xrightarrow{\tau}^*$  denotes the reflexive and transitive closure of the relation  $\xrightarrow{\tau}$ .

### 3 A sort system for DS $\pi$

We now present a novel sort system for the DS $\pi$ -calculus based on Milner's sort system for the polyadic  $\pi$ -calculus [8] and the sort system by Hennessy and

Riely in [10]. In our system we assign sorts to every free identifier and use composite sort expressions as security policies. This lets us avoid the complications introduced by the dependent types used in [10].

Our sort system is likewise capable of controlling whether a certain channel may be used for communication at some location or not. Furthermore it is possible to type the expression  $a!\langle a, b \rangle$  which is not typable in [10].

Under the use of sorts the regulation of migration is refined. Migration of a process from a site  $l$  to the site  $l'$  is allowed whenever site  $l$  is allowed to send code to  $l'$  and  $l'$  is allowed to receive code from  $l$ . By comparison Hennessy and Riely allow migration whenever the destination location has the `go` capability.

The novel part of our sort system is the control of authentication and signing of processes. It is controlled which keys may be used at certain locations and the authentication is only allowed when the authenticated process behaves well. The concept of partial wellsortedness express the fact that a signed process may misbehave under some executions. This will be made clear in the following.

### 3.1 Sorts and sortings

Let  $\mathcal{S}$  where  $\mathcal{S} \cap \mathcal{N} = \emptyset$ , be a countable set of so called ground sorts. A sort system assumes the existence of a *sorting*, which can be viewed a partition of the set of identifiers into equivalence classes called *sorts*. A sorting is a function that maps *sort variables* from the set  $\mathcal{S}$  to sort expressions defined below (where  $\tilde{T}$  denote tuples of sorts).

**Definition 3.** *The set of sort expressions is defined by the grammar:*

$T ::=$	$\text{chan}(\tilde{T})$	<i>Channel sort</i>
	$\text{loc}(\tilde{T})$	<i>Site sort</i>
	$\text{key}(\tilde{T})$	<i>Key sort</i>
	$S$	<i>Sort variable</i>

where  $S \in \mathcal{S}$ . A sort expression which is not a ground sort is called *composite*. The set of composite sort expressions is denoted  $\mathcal{C}$ .

An identifier with the sort constructor  $\text{chan}(T_1, \dots, T_n)$  is an identifier which can be used for input and output of  $n$ -ary tuples where the  $i$ -th component of message should have sort  $T_i$ . A key with the sort constructor  $\text{key}(\tilde{T})$  is a name which can be used as a signature that allows authentication to locations with names whose sort is among the sorts in  $\tilde{T}$ . A location with the sort constructor  $\text{loc}(\tilde{T})$  allows channels whose sort is among the sorts in  $\tilde{T}$  and trusts locations whose sort is in  $\tilde{T}$ . Furthermore only keys whose sort is in  $\text{loc}(\tilde{T})$  are allowed to be used at the location.

A sorting context assigns sorts to free identifiers. If a site  $l$  contains the channel  $a$  we write that name as  $a_l$ . This means that the ground sorts of identifiers are relative to a location.

**Definition 4.** A *sorting context* is a map

$$\Gamma : \mathcal{I} \cup \{v_w \mid v, w \in \mathcal{I}\} \rightarrow \mathcal{S}$$

from identifiers and located identifiers to sort variables. In order for a sorting context to be well defined we demand that the domain of  $\Gamma$  is a finite set. An identifier  $v \in \mathcal{N}$  is said to have the sort  $T$  written  $v : T$  if  $\Gamma(v) = T$  for a fixed context  $\Gamma$ .

$\Gamma, v_w : T$  is the updated sorting context where the identifier  $v_w$  is mapped to sort  $T$ , replacing any possible sort assigned to  $v_w$  in  $\Gamma$ .

**Definition 5.** A *sorting* is a mapping  $\Delta : \mathcal{S} \rightarrow \mathcal{C}$ . We use the notation  $s(v_w)$  for  $\Delta(\Gamma(v_w))$ , i.e. we look up the sorting defined for  $v_w$ .

### 3.2 Type judgements

Our type judgements describe when processes and networks are well-sorted. As mentioned processes and networks can be either *well-sorted* or *partially well-sorted*. A network is partially well-sorted whenever it contains a signed process which may authenticate to locations where it will not be well-sorted.

The type judgments for networks are on the form  $\Gamma \vdash^\Delta M : \diamond$  ( $M$  is well-sorted) and  $\Gamma \vdash^\Delta M : \star$  ( $M$  is partially well-sorted). The sorting judgments for processes are on the form  $\Gamma \vdash_w^\Delta P : \alpha$  where  $w$  is a location and  $\alpha \in \{\star, \diamond\}$  have the same meaning as before.

A network which is partially well sorted may become well sorted. Consider

$$M = l\{\{P\}_k \mid \mathbf{auth}_{\{m,n\}}(l_1, l_2)\}_\circ$$

where we assume that  $P$  contains a subexpression that makes network  $M$  partially well sorted. Since the key  $k$  is not among the keys  $\{m, n\}$  process  $P$  is sandboxed at location  $l_2$ . When  $P$  is placed in  $l_2$ , the network becomes well sorted.

The typing rules are given in Table 5 and 6. Rule (S-OUT) has two side conditions. First, each of the names  $v_i$  which is transmitted should be allowed on the channel  $v$ . Secondly, the channel name  $v$  must be an allowed name at location  $w$ . Rule (S-IN) is quite standard for sort systems for  $\pi$ -calculus; here we also check that the channel  $v$  is permitted at location  $w$ .

To type a parallel composition in (S-PAR), we find the greatest lower bound of the well-sortedness assertions. That is, if some part of the system fails to be well-sorted, the entire system can at most be partially well-sorted.

**Definition 6.** Define an operator  $\sqcap$  on the set  $\{\star, \diamond\}$  by

$$\begin{aligned} \alpha \sqcap \beta &= \diamond, \text{ if } \alpha = \beta = \diamond \\ \alpha \sqcap \beta &= \star, \text{ if } \alpha = \star \text{ or } \beta = \star \end{aligned}$$

In other words  $\sqcap$  is the greatest lower bound with respect to the ordering  $\star \leq \diamond$  on the set  $\{\star, \diamond\}$ .

In (S-Go) we check whether the migrating process  $P$  is well-sorted at its new location  $v$  or not by the demand that  $\Gamma \vdash_v^\Delta P : \diamond$ . Furthermore, migration is only allowed when the source and target location trust each another, i.e when their sortings each contain the site sort of the other location. In the case of migration to a closed location, well-sortedness is guaranteed.

In the rules for signed processes, the sorting of keys is important. The sorts in the sorting for key  $k$  specify the sorts of locations that  $P$  may arrive at. In (S-SIGN1), we therefore require that  $P$  is well sorted at *all* locations whose sort can be found in  $\tilde{T}$ . If this is the case then everything is fine, since  $\{P\}_v$  either authenticates to one of these locations or to a closed location.

The rule (S-SIGN2) describes how a signed process  $\{P\}_v.Q$  may become partially well sorted. This is the case if there exists a location, with sort in the sorting for the key, where process  $P$  is not well sorted. The rule has a negative premise  $\exists u : \Gamma(u) \in \tilde{T} : \Gamma \not\vdash_u^\Delta P : \diamond$ . However, the truth value of this negative premise can always be determined, since it depends only on the subset of the rules defining well-sortedness, namely the set of all rules but (S-SIGN2). These rules all have positive premises only and do not involve judgements of partial well-sortedness. Moreover, there are only finitely many locations  $u$  which have to be checked.

The rule (S-AUTH) checks that the sort of every key in  $K$  contains the sort of  $v$ . In other words all keys in  $K$  should permit authentication to the location  $v$ . Furthermore keys  $K$  used for authentication should be allowed by the location sort  $s(w)$ . Note that we do not demand anything with respect to the location  $v'$ . This is due to the semantics which ensures that the location  $v'$  is a sandbox.

Rule (S-BOX) states that all sandboxes are well sorted – no matter what they contain. This is because migration and authentication of processes are semantically impossible.

### 3.3 Properties of the sort system

Whenever a network obeys the sorting it will keep obeying the sorting no matter how it is evaluated in the semantics.

**Theorem 2 (Subject reduction).** *If  $\Gamma \vdash^\Delta M : \diamond$  and  $M \rightarrow N^*$  then  $\Gamma \vdash^\Delta N : \diamond$*

When a network  $M$  is only partially well-sorted we describe conditions under which  $M$  does not become untypable under evaluation.

We say that all keys used for signing are *unknown* in a given network  $M$  if keys  $k$  used in actions involving signed processes  $\{P\}_k$  are unknown in all corresponding authentication prefixes  $\mathbf{auth}_K(l_1, l_2)$ , i.e.  $k \notin K$ . If this property holds for all signed processes in a network then all keys are said to be unknown.

**Definition 7.** *All keys of signed processes are said to be unknown in a network  $M$  if*

$$\forall l : M \xrightarrow{\{P\}_k @ l} M' \Rightarrow \forall M \xrightarrow{\mathbf{auth}_K(l_1, l_2) @ l} M'' : k \notin K \wedge (K \cup \{k\}) \cap \mathcal{V} = \emptyset$$

---

(S-NIL-PROC) $\frac{}{\Gamma \vdash_w^{\Delta} 0 : \diamond}$	
(S-PAR) $\frac{\Gamma \vdash_w^{\Delta} P : \alpha \quad \Gamma \vdash_w^{\Delta} Q : \beta}{\Gamma \vdash_w^{\Delta} P \mid Q : \alpha \sqcap \beta}$	
(S-BANG) $\frac{\Gamma \vdash_w^{\Delta} P : \alpha}{\Gamma \vdash_w^{\Delta} *P : \alpha}$	(S-NEW1) $\frac{\Gamma, n_w : T \vdash_w^{\Delta} P : \alpha}{\Gamma \vdash_w^{\Delta} (\nu n : T) P : \alpha}$
(S-OUT) $\frac{\Gamma \vdash_w^{\Delta} P : \alpha}{\Gamma \vdash_w^{\Delta} v!(v_1, \dots, v_n).P : \alpha},$	$\left( \begin{array}{l} s(v_w) = \mathbf{chan}(T_1, \dots, T_n) \\ \forall 1 \leq i \leq n : T_i = \Gamma(v_{i_w}) \\ \mathbf{loc}(\tilde{T}) = s(w), \Gamma(v_w) \in \tilde{T} \end{array} \right)$
(S-IN) $\frac{\Gamma, x_{1_w} : T_1, \dots, x_{n_w} : T_n \vdash_w^{\Delta} Q : \alpha}{\Gamma \vdash_w^{\Delta} v?(x_1, \dots, x_n).Q : \alpha}$	$\left( \begin{array}{l} T = \mathbf{chan}(T_1, \dots, T_n) \\ \mathbf{loc}(\tilde{T}) = s(w), \Gamma(v_w) \in \tilde{T} \end{array} \right)$
(S-GO) $\frac{\Gamma \vdash_w^{\Delta} P : \alpha}{\Gamma \vdash_w^{\Delta} \mathbf{go} \ v_{\gamma}.P : \alpha}$	$\left( \begin{array}{l} \mathbf{loc}(\tilde{T}_v) = s(v) \\ \mathbf{loc}(\tilde{T}_w) = s(w) \\ (\gamma = \bullet \vee (\Gamma(w) \in \tilde{T}_v \wedge \Gamma(v) \in \tilde{T}_w)) \end{array} \right)$
(S-MATCH) $\frac{\Gamma \vdash_w^{\Delta} P : \alpha}{\Gamma \vdash_w^{\Delta} [\tilde{u}\sqrt{v}] P : \alpha},$	$\sqrt{\in} \in \{=, \neq\}$
(S-SIGN1) $\frac{\forall u : \Gamma(u) \in \tilde{T} : \Gamma \vdash_u^{\Delta} P : \diamond \quad \Gamma \vdash_w^{\Delta} Q : \alpha, s(v_w) = \mathbf{key}(\tilde{T})}{\Gamma \vdash_w^{\Delta} \{P\}_v.Q : \alpha}$	
(S-SIGN2) $\frac{\exists u : \Gamma(u) \in \tilde{T} : \Gamma \vdash_u^{\Delta} P : \diamond \quad \Gamma \vdash_w^{\Delta} Q : \alpha, s(v_w) = \mathbf{key}(\tilde{T})}{\Gamma \vdash_w^{\Delta} \{P\}_v.Q : \star}$	
(S-AUTH) $\frac{\Gamma \vdash_w^{\Delta} P : \alpha}{\Gamma \vdash_w^{\Delta} \mathbf{auth}_K(v, v').P : \alpha},$	$\left( \forall k \in K : \mathbf{key}(\tilde{T}_k) = s(k_w) \wedge \Gamma(v) \in \tilde{T}_k \wedge \right)$ $\mathbf{loc}(\tilde{T}) = s(w) \wedge \Gamma(k_w) \in \tilde{T}$

---

**Table 5.** The typing rules for  $S$

---

(S-NIL-NET) $\frac{}{\Gamma \vdash^{\Delta} 0 : \diamond}$	(S-SITE) $\frac{\Gamma \vdash_l^{\Delta} P : \alpha}{\Gamma \vdash^{\Delta} l[P]_{\circ} : \alpha}$
(S-BOX) $\frac{}{\Gamma \vdash^{\Delta} l[P]_{\bullet} : \diamond}$	
(S-NEW2) $\frac{\Gamma, n_l : T \vdash^{\Delta} N : \alpha}{\Gamma \vdash^{\Delta} (\nu_l n : T) N : \alpha}$	(S-PAR-NET) $\frac{\Gamma \vdash^{\Delta} M : \alpha \quad \Gamma \vdash^{\Delta} N : \beta}{\Gamma \vdash^{\Delta} M \mid N : \alpha \sqcap \beta}$

---

**Table 6.** The typing rules for networks  $S$

**Theorem 3 (Weak subject reduction).** *If  $\Gamma \vdash^\Delta M : \star$  and  $M \xrightarrow{\tau} M'$  and all keys used for signing are unknown then  $M'$  is typable i.e.  $\Gamma \vdash^\Delta M' : \diamond$  or  $\Gamma \vdash^\Delta M' : \star$ .*

*Remark 1.* A network which is partially well sorted with unknown keys may never become well sorted. Consider the network

$$M = l[\{P\}_k \mid a(x).\mathbf{auth}_{\{m,n\}}(l_1, l_2)]_\circ$$

where we assume that  $P$  contains a subexpression that makes network  $M$  partially well sorted. Since synchronization never occurs on channel  $a$  process  $P$  will never be sandboxed – hence  $M$  will remain partially well sorted.

In order to describe the errors detected by the sort system we define an error predicate on networks. The predicate  $\xrightarrow{\text{err}}$  is defined relative to a sort context  $\Gamma$  and a sorting  $\Delta$  (see table 7). The rules (E-OUT) and (E-IN) describe unauthorized use of communications channels or the use of a channel in an unsorted way. Rule (E-GO) describes a migration where process  $P$  is either untrusted at the destination site or unallowed to go from the source site. Rule E-Auth describes an unauthorized use of a key at an open location.

<p>(E-OUT) <math>l[a!(v_1, \dots, v_n).Q]_\circ \xrightarrow{\text{err}}_{\Gamma, \Delta}</math> if <math>\Gamma(a_i) \notin s(l)</math> or <math>\exists i : \Gamma(v_{i_l}) \neq T_i</math>  where <math>\mathbf{chan}(T_1, \dots, T_n) = s(a_l)</math></p> <p>(E-IN) <math>l[a?(x_1, \dots, x_n : T).Q]_\circ \xrightarrow{\text{err}}_{\Gamma, \Delta}</math> if <math>\Gamma(a_l) \notin s(l)</math> or <math>\exists i : \Gamma(x_{i_l}) \neq T_i</math>  where <math>\mathbf{chan}(T_1, \dots, T_n) = T</math></p> <p>(E-GO) <math>l[\mathbf{go} \ u_\circ.P]_\circ \xrightarrow{\text{err}}_{\Gamma, \Delta}</math> if <math>\Gamma(l) \notin s(u_l)</math> or <math>\Gamma(u_l) \notin s(l)</math></p> <p>(E-AUTH) <math>l[\mathbf{auth}_K(l_1, l_2).P]_\circ \xrightarrow{\text{err}}_{\Gamma, \Delta}</math> if <math>\exists k \in K : \Gamma(k_l) \notin s(l)</math></p>	<p>(E-PAR) <math>\frac{M \xrightarrow{\text{err}}_{\Gamma, \Delta}}{M \mid N \xrightarrow{\text{err}}_{\Gamma, \Delta}}</math></p> <p>(E-STRUCT) <math>\frac{N \xrightarrow{\text{err}}_{\Gamma, \Delta} \quad M \equiv N}{M \xrightarrow{\text{err}}_{\Gamma, \Delta}}</math></p> <p>(E-RES) <math>\frac{M \xrightarrow{\text{err}}_{(\Gamma, n_l : T), \Delta}}{(\nu_l \ n : T)M \xrightarrow{\text{err}}_{\Gamma, \Delta}}</math></p> <p>Otherwise <math>\not\xrightarrow{\text{err}}_{\Gamma, \Delta}</math></p>
---	--

**Table 7.** The error predicate defined on networks

**Definition 8.** *A network  $N$  is said to error for a given context  $\Gamma$  and sorting  $\Delta$  if it holds that  $N \xrightarrow{\text{err}}_{\Gamma, \Delta}$ .*

The following theorem states that a well-sorted network will not exhibit any of the errors described by the error predicate.

**Theorem 4 (Type safety).** *If  $\Gamma \vdash^\Delta M : \diamond$  then  $M \not\rightarrow_{\Gamma, \Delta}^{\text{err}}$ .*

Combining the results of subject reduction and type safety we get that a well-sorted network will never produce a violation of the security policy along any path of execution.

**Corollary 1.** *If  $\Gamma \vdash^\Delta M : \diamond$  and  $M \rightarrow^* M'$  then  $M' \not\rightarrow_{\Gamma, \Delta}^{\text{err}}$ .*

*Example 3 (Control of system resources).* We revisit Example 2. Inspect the system given file system below. Recall the creation of files

$$\text{CreateFile}(n, c) \stackrel{\text{def}}{=} *req?(s, fname).[fname = n]go\ s_{\circ}.io!\langle c \rangle$$

and the file system

$$\begin{aligned} \text{IOSystem} &\stackrel{\text{def}}{=} \\ &\text{Files} \quad []_{\circ} \mid \\ &\text{ReadFile} \quad [*req?(from, fname).go\ \text{Files}_{\circ}.req!\langle from, fname \rangle]_{\circ} \mid \\ &\text{WriteFile} \quad [*req?(n, c).go\ \text{Files}_{\circ}.CreateFile(n, c)]_{\circ} \end{aligned}$$

Again we define a network  $M$  which tries to create a file with file name  $myFile$  and content  $a$ .

$$M \stackrel{\text{def}}{=} \text{IOSystem} \mid l[go\ \text{WriteFile}_{\circ}.req!\langle myFile, a \rangle]_{\circ}$$

In order to allow this file writing we choose the following sort context (where the ground sorts and the sorting for IOSystem are omitted)

$$\Gamma = \{l \mapsto \lambda, \text{WriteFile}_l \mapsto w, req_l \mapsto \rho, myFile_l \mapsto \mu, a_l \mapsto \alpha, \dots\}$$

and sorting

$$\Delta = \{\lambda \mapsto \text{loc}(\{w, \dots\}), w \mapsto \text{loc}(\{\lambda, \dots\}), \rho \mapsto \text{chan}(\mu, \alpha), \dots\}.$$

The file can be written because the sites trust each other with respect to migration. If on the other hand we want to disallow the writing of files this can be done in two ways. We can define a sorting  $\Delta = \{\lambda \mapsto \text{loc}(), w \mapsto \text{loc}(\{\lambda, \dots\}), \dots\}$  which states that the site  $l$  do not allow the writing of files (i.e. migration to the file writing demon). We can also define a sorting  $\Delta = \{\lambda \mapsto \text{loc}(\{w, \dots\}), w \mapsto \text{loc}(), \dots\}$  which can be interpreted as that the write demon do not trust any users who attempt to write files.

## 4 Conclusions and future work

We have developed an extension of the  $D\pi$ -calculus of Riely et al. called the  $DS\pi$ -calculus, in which security problems related to authentication and migration have been studied. Our main interest has been to discover how authentication of new processes can be controlled under the use of a sort system and a

distinction between open and closed sites. The semantics of closed sites does not allow migration or authentication of new processes, thus capturing the notion of sandboxing.

We have described a sort system which ensures that both communication and the use of keys happen correctly on open locations. Furthermore migration is only allowed when the involved sites trust each other. Under the use of the labelled semantics for  $DS\pi$  we have shown a weak subject reduction result – partially well sorted networks may become well sorted, if all keys used for signing processes are unknown. This idea as well as the use of sorts for the control of authentication appear to be novel.

The semantics of sandboxes can easily be refined. It would be interesting to tag each site with a set of rights or even with a sorting – creating a semantics which could determine whether rights were violated or not for sand boxes using dynamic typing. The sort system would then be in charge of ensuring safety for the un-sandboxed rest of the network.

It would also be interesting to consider the introduction of subtyping into the sort system, as this would allow us to distinguish between keys and sites operating at different levels of security. In this setting, a sandbox would be at the lowest level of security.

Finally, one should investigate the expressive power of the  $DS\pi$ -calculus compared to  $D\pi$ . It appears straightforward to encode the prefixes  $\{P\}_v$ , and  $\text{auth}_K(v_1, v_2)$  within a version of  $D\pi$  with match and mismatch. However, the encoding of open and closed locations appears to require a notion of blocking [13].

## References

1. Luca Cardelli and Andrew D. Gordon. Mobile Ambients. Foundations of Software Science and Computation Structures: First International Conference, FOSSACS '98.
2. Michele Bugliesi and Silvia Crafa and Amela Prelic and Vladimiro Sassone. Secrecy in untrusted networks. In Proceedings of 30th International Colloquium on Automata, Languages and Programming, ICALP 2003.
3. Mario Coppo and Mariangiola Dezani-Ciancaglini and Elio Giovannetti and Rosario Pugliese: Dynamic and Local Typing for Mobile Ambients. IFIP TCS. pp. 577–590, 2004.
4. Daniele Gorla, Matthew Hennessy and Vladimiro Sassone. Security Policies as Membranes in Systems for Global Computing. Electr. Notes Theor. Comput. Sci., Vol. 138, No. 1, 2005.
5. James Riely and Matthew Hennessy. A typed language for distributed mobile processes (extended abstract). POPL '98: Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages. pp. 378–390, 1998.
6. Tim Lindholm and Fran Yellin: The Java™ Virtual Machine Specification, Second Edition, Addison-Wesley, 1999.
7. Massimo Merro and Matthew Hennessy: Bisimulation congruences in safe ambients, Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pp. 71–80, 2002.

8. Robin Milner: The polyadic pi-calculus: a tutorial in Logic and Algebra of Specification, Springer-Verlag, editors F. L. Bauer and W. Brauer and H. Schwichtenberg, pp. 203–246, 1993.
9. Benjamin C. Pierce and Davide Sangiorgi. Typing and Subtyping for Mobile Processes. In *Proceedings of 8th Annual IEEE Symposium on Logic in Computer Science*, 1993, pp. 376–385. Full version in *Mathematical Structures in Computer Science*, Vol. 6, No. 5, 1996.
10. James Riely and Matthew Hennessy: Resource Access Control in Systems of Mobile Agents. *Electronic Notes in Theoretical Computer Science*. Vol 16, 1998
11. Davide Sangiorgi and David Walker. *The  $\pi$ -calculus: A Theory of Mobile Processes*, Cambridge University Press 2001.
12. Peter Sewell and Jan Vitek. Secure Composition of Insecure Components. *Proceedings of The 12th Computer Security Foundations Workshop*, pp. 136–150, 1999.
13. J. L. Vivas. *Dynamic Binding of Names in Calculi for Mobile Processes*. Ph.D. thesis, Royal Institute of Technology, Sweden, 2001.