Secrecy and Group Creation

Luca Cardelli [1] Giorgio Ghelli [2] Andrew D. Gordon [1]

Abstract

We add an operation of group creation to the typed π-calculus, where a group is a type for channels. Creation of fresh groups has the effect of statically preventing certain communications, and can block the accidental or malicious leakage of secrets. Intuitively, no channel belonging to a fresh group can be received by processes outside the initial scope of the group, even if those processes are untyped. We formalize this intuition by adapting a notion of secrecy introduced by Abadi, and proving a preservation of secrecy property.

Key words: π-calculus, secrecy, security types

1 Introduction

Group creation is a natural extension of the sort-based type systems developed for the π-calculus. However, group creation has an interesting and subtle connection with secrecy. We start from the untyped π-calculus, where an operation to create fresh communication channels can be interpreted as creating fresh secrets. Under this interpretation, though, secrets can be leaked. We then introduce the notion of groups, which are types for channels, together with an operation for creating fresh groups. We explain how a fresh secret belonging to a fresh group can never be communicated to anybody who does not know the group in the first place. In other words, our type system prevents secrets from being leaked. Crucially, groups are not values, and cannot be communicated; otherwise, this secrecy property would fail.

[1] Microsoft Research, Roger Needham Building, 7 Thomson Ave, Cambridge, UK
[2] Università di Pisa, Dipartimento di Informatica, Via Buonarroti 2, Pisa, Italy

1.1 Leaking Secrets

Consider the following configuration, where P is a private subsystem (a player) running in parallel with a potentially hostile adversary O (an opponent).

$$O \mid P$$

Suppose that the player P wants to create a fresh secret x. For example, x could be a private communication channel to be used only between subsystems of P. In the π-calculus this can be done by letting P evolve into a configuration $(\nu x)P'$, which means: create a new channel x to be used in the scope of P'.

$$O \mid (\nu x)P'$$

The channel x is intended to remain private to P'. This privacy policy is going to be violated if the system then evolves into a situation such as the following, where p is a public channel known to the opponent ($p(y)$ is input of y on p, and $\overline{p}\langle x\rangle$ is output of x on p):

$$p(y).O' \mid (\nu x)(\overline{p}\langle x\rangle \mid P'')$$

In this situation, the name x is about to be sent by the player over the public channel p and received by the opponent. In order for this communication to happen, the rules of the π-calculus, described in Section 2, require first an enlargement (extrusion) of the scope of x (otherwise x would escape its lexical scope). We assume that x is different from p, y, and any other name in O', so that the enlargement of the scope of x does not cause name conflicts. After extrusion, we have:

$$(\nu x)(p(y).O' \mid \overline{p}\langle x\rangle \mid P'')$$

Now, x can be communicated over p into the variable y, while keeping x entirely within the scope of (νx). This results in:

$$(\nu x)(O'\{y{\leftarrow}x\} \mid P'')$$

where the opponent has acquired the secret.

The private name x has been leaked to the opponent by a combination of two mechanisms: the output instruction $\overline{p}\langle x\rangle$, and the extrusion of (νx). Can we prevent this kind of leakage of information? We have to consider that such a leakage may arise simply because of a mistake in the code of the player P, or because P decides to violate the privacy policy of x, or because a subsystem of P acts as a spy for the opponent.

It seems that we need to restrict either communication or extrusion. Since names are dynamic data in the π-calculus, it is not easy to say that a situation such as $\overline{p}\langle x\rangle$ (sending x on a channel known to the opponent) should not arise, because p may be dynamically obtained from some other channel, and may not occur at all in the code of P.

The other possibility is to try to prevent extrusion, which is a necessary step when leaking names outside their initial scope. However, extrusion is a fundamental mechanism in the π-calculus: blocking it completely would also block innocent communications over p. In general, attempts to limit extrusion are problematic, unless we abandon the notion of "fresh channel" altogether.

A natural question is whether one could somehow declare x to be private, and have this assertion statically checked so that the privacy policy of x cannot be violated. To this end, we may consider typed versions of the π-calculus. In these systems, we can classify channels into different groups (usually called sorts in the literature). We could have a group G for our private channels and write $(\nu x{:}G)P'$ to declare x to be of sort G. Unfortunately, in standard π-calculus type systems all the groups are global, so the opponent could very well mention G in an input instruction. Global groups do not offer any protection, because leakage to the opponent can be made to typecheck:

$$p(y{:}G).O' \mid (\nu x{:}G)(\overline{p}\langle x\rangle \mid P'')$$

In order to guarantee secrecy, we would want the group G itself to be secret, so that no opponent can input names of group G, and that no part of the player can output G information on public channels. A first idea is to partition groups into public ones and secret ones, with the static constraints that members of secret groups cannot be communicated over channels of public groups [9]. But this would work only for systems made of two (or a fixed number of) distrustful components; we aim to find a more general solution.

1.3 Group Creation

In general, we want the ability to create fresh groups on need, and then to create fresh elements of those groups. To this end, we extend the π-calculus with an operator, $(\nu G)P$, to dynamically create a new group G in a scope P. This is a dynamic operator because, for example, it can be used to create a fresh group after an input:

$$q(y{:}T).(\nu G)P$$

Although group creation is dynamic, the group information can be tracked statically to ensure that names of different groups are not confused. Moreover, dynamic group creation can be very useful: we can dynamically spawn subsystems that have their own pool of shared resources that cannot interfere with other subsystems (compare with applet sandboxing).

Our troublesome example can now be represented as follows, where G is a new group, $G[\,]$ is the type of channels of group G, and a fresh x is declared to be a channel of group G (the type structure will be explained in more detail later):

$$p(y{:}T).O' \mid (\nu G)(\nu x{:}G[\,])\overline{p}\langle x\rangle$$

Here an attempt is made again to send the channel x over the public channel p. Fortunately, this process cannot be typed: the type T would have to mention G, in order to receive a channel of group G, but this is impossible because G is not known in the global scope where p would have to have been declared. The construct (νG) has extrusion properties similar to (νx), which are needed to permit legal communications over channels unrelated to G channels, but these extrusion rules prevent G from being confused with any group mentioned in the type T.

1.4 Untyped Opponents

Let us now consider the case where the opponent is untyped or, equivalently, not well-typed. This is intended to cover the situation where an opponent can execute any instruction available in the computational model without being restricted by static checks such as typechecking or bytecode verification. For example, the opponent could be running on a separate, untrusted, machine.

We first make explicit the type declaration of the public channel, $p{:}U$, which had so far been omitted. The public channel must have a proper type, because that type is used in checking the type correctness of the player, at least. This type declaration could take the form of a channel declaration $(\nu p{:}U)$ whose scope encloses both the player and the opponent, or it could be part of some declaration environment shared by the player and the opponent and provided by a third entity in the system (e.g., a name server).

Moreover, we remove the typing information from the code of the opponent, since an opponent does not necessarily play by the rules. The opponent now attempts to read any message transmitted over the public channel, no matter what its type is.

$$(\nu p{:}U)(\ \dots\ p(y).O' \mid (\nu G)(\nu x{:}G[\,])\overline{p}\langle x\rangle)$$

Will an untyped opponent, by cheating on the type of the public channel, be able to acquire secret information? Fortunately, the answer is still no. The fact that the player is well-typed is sufficient to ensure secrecy, even in the presence of untyped opponents. This is because, in order for the player to leak information over a public channel p, the output operation $\overline{p}\langle x\rangle$ must be well-typed. The name x can be communicated only on channels whose type mentions G. So the output $\overline{p}\langle x\rangle$ cannot be well-typed, because then the type U of p would have to mention the group G, but U is not in the scope of G.

The final option to consider is whether one can trust the source of the declaration $p{:}U$. This declaration could come from a trusted source distinct from the opponent, but in general one has to mistrust this information as well. In any case, we can assume that the player will be typechecked with respect to this questionable information, $p{:}U$, within a trusted context. Even if U tries to cheat by mentioning G, the typing rules will not confuse that G with the one occurring in the player as (νG), and the output operation $\overline{p}\langle x\rangle$ will still fail to typecheck. The only important requirement is that the player must be typechecked with respect to a global environment within a trusted context, which seems reasonable. This is all our secrecy theorem (Section 3) needs to assume.

1.5 Secrecy

We have thus established, informally, that a player creating a fresh group G can never communicate channels of group G to an opponent outside the initial scope of G, either because a (well-typed) opponent cannot name G to receive the message, or, in any case, because a well-typed player cannot use public channels to communicate G information to an (untyped) opponent:

Channels of group G are forever secret
outside the initial scope of (νG).

So, secrecy is reduced in a certain sense to scoping and typing restrictions. But the situation is fairly subtle because of the extrusion rules associated with scoping, the fact that scoping restrictions in the ordinary π-calculus do not prevent leakage, and the possibility of untyped opponents. As we have seen, the scope of channels can be extruded too far, perhaps inadvertently, and cause leakage, while the scope of groups offers protection against accidental or malicious leakage, even though it can be extruded as well.

We organise the remainder of the paper as follows. Section 2 defines the syntax, reduction semantics, and type system of our typed π-calculus with groups. In Section 3 we present Abadi's notion of secrecy in terms of the untyped π-calculus. We also state the main technical result of the paper, Theorem 1, that a well-typed process preserves the secrecy of a fresh name of a fresh group, even from an untyped opponent. We outline the proof of Theorem 1 in Section 4; the main idea of the proof is to separate trusted data (from the typed process) and untrusted data (from the untyped opponent) using an auxiliary type system defined on untyped processes. Finally, Section 5 concludes. Appendixes contain proofs omitted from the body of the paper.

A preliminary version of part of this work appears as a conference paper [6].

2 A Typed π-Calculus with Groups

We present here a typed π-calculus with groups and group creation. Milner's sort system [13,14] is the earliest type system for the π-calculus. Sorts are like groups in that each name belongs to a sort, but Milner's system has no construct for sort creation. In our calculus, a replicated process $!(\nu G)(\nu x{:}G[T])(P \mid Q)$ makes an unbounded number of copies of the pair of processes P and Q, and our type system guarantees each pair exclusive access to a fresh channel x. Such exclusion does not follow from Milner's system, as it is limited to a fixed set of sorts. On the other hand, his system allows recursive definitions of sorts; we would need to add recursive types to our system to mimic such definitions. Subsequent type systems introduce a variety of channel type constructors and subtyping [15,16].

2.1 Syntax and Operational Semantics

Types specify, for each channel, its group and the type of the values that can be exchanged on that channel.

Types:

$T ::=$	channel type
$\quad G[T_1, \ldots, T_n]$	$\quad$ polyadic channel in group G

We study an asynchronous, choice-free, polyadic typed π-calculus. The calculus is defined as follows. We identify processes up to capture-avoiding renaming of bound variables.

Expressions and Processes:

x, y, p, q	names, variables
$P, Q, R ::=$	process
$\quad x(y_1{:}T_1, \ldots, y_k{:}T_k).P$	$\quad$ channel input
$\quad \overline{x}\langle y_1, \ldots, y_k\rangle$	$\quad$ channel output
$\quad (\nu G)P$	$\quad$ group creation
$\quad (\nu x{:}T)P$	$\quad$ restriction
$\quad P \mid Q$	$\quad$ composition
$\quad !P$	$\quad$ replication
$\quad \mathbf{0}$	$\quad$ inactivity

In a restriction, $(\nu x{:}T)P$, the name x is bound in P, and in an input, $x(y_1{:}T_1, \ldots, y_k{:}T_k).P$, the names $y_1, \ldots, y_k$ are bound in P. In a group creation $(\nu G)P$, the group G is bound with scope P. Let $fn(P)$ be the set the names free in a process P, and let $fg(P)$ and $fg(T)$ be the sets of groups free in a process P and a type T, respectively.

In the next two tables, we define a reduction relation $P \rightarrow Q$ in terms of an auxiliary notion of structural congruence $P \equiv Q$. Structural congruence allows a process to be re-arranged so that reduction rules may be applied. Each reduction derives from an exchange of a tuple on a named channel.

Our rules for reduction and structural congruence are standard [14] apart from the inclusion of new rules for group creation, and the exclusion of the usual garbage collection rules such as $\mathbf{0} \equiv (\nu x{:}T)\mathbf{0}$ or $x \notin fn(P) \Rightarrow (\nu x{:}T)P \equiv P$. Such rules are unnecessary for calculating reduction steps; their inclusion would not create major problems, but they would slightly complicate the statement of subject reduction.

Structural Congruence: $P \equiv Q$

$P \equiv P$	(Struct Refl)
$Q \equiv P \Rightarrow P \equiv Q$	(Struct Symm)
$P \equiv Q, Q \equiv R \Rightarrow P \equiv R$	(Struct Trans)
$P \equiv Q \Rightarrow (\nu x{:}T)P \equiv (\nu x{:}T)Q$	(Struct Res)

$P \equiv Q \Rightarrow (\nu G)P \equiv (\nu G)Q$ (Struct GRes)
$P \equiv Q \Rightarrow P \mid R \equiv Q \mid R$ (Struct Par)
$P \equiv Q \Rightarrow\, !P \equiv\, !Q$ (Struct Repl)
$P \equiv Q \Rightarrow x(y_1{:}T_1, \ldots, y_n{:}T_n).P \equiv x(y_1{:}T_1, \ldots, y_n{:}T_n).Q$ (Struct Input)

$P \mid \mathbf{0} \equiv P$ (Struct Par Zero)
$P \mid Q \equiv Q \mid P$ (Struct Par Comm)
$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$ (Struct Par Assoc)
$!P \equiv P \mid\, !P$ (Struct Repl Par)

$x_1 \neq x_2 \Rightarrow (\nu x_1{:}T_1)(\nu x_2{:}T_2)P \equiv (\nu x_2{:}T_2)(\nu x_1{:}T_1)P$ (Struct Res Res)
$x \notin \mathit{fn}(P) \Rightarrow (\nu x{:}T)(P \mid Q) \equiv P \mid (\nu x{:}T)Q$ (Struct Res Par)
$(\nu G_1)(\nu G_2)P \equiv (\nu G_2)(\nu G_1)P$ (Struct GRes GRes)
$G \notin \mathit{fg}(T) \Rightarrow (\nu G)(\nu x{:}T)P \equiv (\nu x{:}T)(\nu G)P$ (Struct GRes Res)
$G \notin \mathit{fg}(P) \Rightarrow (\nu G)(P \mid Q) \equiv P \mid (\nu G)Q$ (Struct GRes Par)

Reduction: $P \rightarrow Q$

$\overline{x}\langle y_1, \ldots, y_n\rangle \mid x(z_1{:}T_1, \ldots, z_n{:}T_n).P \rightarrow P\{y_1{\leftarrow}z_1\} \cdots \{y_n{\leftarrow}z_n\}$ (Red I/O)
$P \rightarrow Q \Rightarrow P \mid R \rightarrow Q \mid R$ (Red Par)
$P \rightarrow Q \Rightarrow (\nu G)P \rightarrow (\nu G)Q$ (Red GRes)
$P \rightarrow Q \Rightarrow (\nu x{:}T)P \rightarrow (\nu x{:}T)Q$ (Red Res)
$P' \equiv P, P \rightarrow Q, Q \equiv Q' \Rightarrow P' \rightarrow Q'$ (Red $\equiv$)

The new rules for group creation are the congruence rules (Struct GRes) and (Red GRes) and the scope mobility rules are (Struct GRes GRes), (Struct GRes Res), and (Struct GRes Par). The latter rules are akin to the standard scope mobility rules for restriction, (Struct Res Res) and (Struct Res Par).

2.2 The Type System

Environments declare the names and groups in scope during type-checking; we define environments, E, by $E ::= \varnothing \mid E, G \mid E, x{:}T$. We define $dom(E)$ by $dom(\varnothing) = \varnothing$, $dom(E, G) = dom(E) \cup \{G\}$, and $dom(E, x{:}T) = dom(E) \cup \{x\}$.

We define four typing judgments: first, $E \vdash \diamond$ means that E is well-formed, that is, that no name or group appears twice, and that every type in E is well-formed; second, $E \vdash T$ means that every free group in T is defined in E; third, $E \vdash x : T$ means that $x{:}T$ is in E, and that E is well-formed; and, fourth, $E \vdash P$ means that P is well-formed in the environment E.

Throughout the paper, any antecedent of the form $E \vdash \mathbb{J}_1, \ldots, E \vdash \mathbb{J}_n$ means $E \vdash \diamond$ when $n = 0$.

Typing Judgments:

$E \vdash \diamond$	good environment
$E \vdash T$	good channel type T
$E \vdash x : T$	good name x of channel type T
$E \vdash P$	good process P

Typing Rules:

(Env $\varnothing$)

$$\frac{}{\varnothing \vdash \diamond}$$

(Env x)

$$\frac{E \vdash T \quad x \notin \mathit{dom}(E)}{E, x{:}T \vdash \diamond}$$

(Env G)

$$\frac{E \vdash \diamond \quad G \notin \mathit{dom}(E)}{E, G \vdash \diamond}$$

(Type Chan)

$$\frac{G \in \mathit{dom}(E) \quad E \vdash T_1 \quad \cdots \quad E \vdash T_n}{E \vdash G[T_1, \ldots, T_n]}$$

(Exp x)

$$\frac{E', x{:}T, E'' \vdash \diamond}{E', x{:}T, E'' \vdash x : T}$$

(Proc GRes)

$$\frac{E, G \vdash P}{E \vdash (\nu G)P}$$

(Proc Res)

$$\frac{E, x{:}T \vdash P}{E \vdash (\nu x{:}T)P}$$

(Proc Zero)

$$\frac{E \vdash \diamond}{E \vdash \mathbf{0}}$$

(Proc Par)

$$\frac{E \vdash P \quad E \vdash Q}{E \vdash P \mid Q}$$

(Proc Repl)

$$\frac{E \vdash P}{E \vdash !P}$$

(Proc Input)

$$\frac{E \vdash x : G[T_1, \ldots, T_n] \qquad E, y_1{:}T_1, \ldots, y_n{:}T_n \vdash P}{E \vdash x(y_1{:}T_1, \ldots, y_n{:}T_n).P}$$

(Proc Output)

$$\frac{E \vdash x : G[T_1, \ldots, T_n] \qquad E \vdash y_1 : T_1 \quad \cdots \quad E \vdash y_n : T_n}{E \vdash \overline{x}\langle y_1, \ldots, y_n \rangle}$$

The rules for good environments ensure that the names and groups declared in an environment are distinct, and that all the types mentioned in an environment are good. The rule for a good type ensures that all the groups free in a type are declared. The rule for a good name looks up the type of a name in the environment. The rules (Proc Input) and (Proc Output) for well-typed processes ensure that names occurring in inputs and outputs are used according to their declared types. The rules (Proc GRes) and (Proc Res) allow fresh groups and names, respectively, to be used inside their scope but not outside. The other rules (Proc Zero), (Proc Par), and (Proc Repl) define a composite process to be well-typed provided its components, if any, are themselves well-typed.

2.3 Subject Reduction

Subject reduction is a property stating that well-typed processes reduce necessarily to well-typed processes, thus implying that "type errors" are not generated during reduction. As part of establishing this property, we need to establish a subject congruence property, stating that well-typing is preserved by congruence. Subject congruence is essential for a type system based on the π-calculus: two congruent processes are meant to represent the same computation so they should have the same typing properties.

As we shall see shortly, a consequence of our typing discipline is the ability to preserve secrets. In particular, the subject reduction property, together with the proper application of extrusion rules, has the effect of preventing certain communications that would leak secrets. For example, consider the discussion in Section 1.3, regarding the process:

$$p(y{:}T).O' \mid (\nu G)(\nu x{:}G[\,])P$$

In order to communicate the name x (the secret) on the public channel p, we would need to reduce the initial process to a configuration containing the following:

$$p(y{:}T).O'' \mid \overline{p}\langle x\rangle$$

If subject reduction holds then this reduced term has to be well-typed, which is true only if $p : H[T]$ for some H, and $T = G[\,]$. However, in order to get to the point of bringing the input operation of the opponent next to an output operation of the player, we must have extruded the (νG) and $(\nu x{:}G[\,])$ binders outward. The rule (Struct GRes Par), used to extrude (νG) past $p(y{:}T).O''$, requires that $G \notin fg(T)$. This contradicts the requirement that $T = G[\,]$.

We prove the following lemma and proposition in Appendix A.

Lemma 1 (Subject Congruence) *If $E \vdash P$ and $P \equiv Q$ then $E \vdash Q$.*

Proposition 2 (Subject Reduction) *If $E \vdash P$ and $P \rightarrow Q$ then $E \vdash Q$.*

Subject reduction allows us to prove secrecy properties such as the following.

Proposition 3 *Let the process $P = p(y{:}T).O' \mid (\nu G)(\nu x{:}G[T_1, \ldots, T_n])P'$. If $E \vdash P$, for some E, then no process deriving from P includes a communication of x along p. Formally, there is no process P'' and no context $C[\,]$ such that $P \equiv (\nu G)(\nu x{:}G[T_1, \ldots, T_n])P''$, $P'' \rightarrow^* C[\overline{p}\langle x\rangle]$, where p and x are not bound by $C[]$.*

Proof Assume that P'' and $C[\,]$ exist. Subject reduction implies the judgment $E, G, x{:}G[T_1, \ldots, T_n] \vdash C[\overline{p}\langle x\rangle]$, which implies that $E, G, x{:}G[T_1, \ldots, T_n], E' \vdash \overline{p}\langle x\rangle$ for some E', by induction on the size of the context. Hence, p has a type $H[G[T_1, \ldots, T_n]]$. But this is impossible, since p is defined in E, and hence is outside the scope of G. $\square$

In the following section we generalize this result, and extend it to a situation where the opponent is not necessarily well-typed.

3 Secrecy in the Context of an Untyped Opponent

We formalize the idea that in the process $(\nu G)(\nu x{:}G[T_1, \ldots, T_n])P$, the name x of the new group G is known only within P (the scope of G) and hence is kept secret from any opponent able to communicate with the process (whether or not the opponent respects our type system). We give a precise definition of when an untyped process $(\nu x)P$ preserves the secrecy of a restricted name x from an opponent (the external process with which it interacts). Then we show that the untyped process obtained by erasing type annotations and group restrictions from a well-typed process $(\nu G)(\nu x{:}G[T_1, \ldots, T_n])P$ preserves the secrecy of the name x.

3.1 Review: The Untyped π-Calculus

In this section, we describe the syntax and semantics of an untyped calculus that corresponds to the typed calculus of Section 2. The process syntax is the same as for the typed calculus, except that we drop type annotations and the new-group construct.

Processes:

x, y, p, q	names, variables
$P, Q, R ::=$	process
$\quad x(y_1, \ldots, y_n).P$	$\quad$ polyadic input
$\quad \overline{x}\langle y_1, \ldots, y_n\rangle$	$\quad$ polyadic output

$(\nu x)P$	restriction
$P \mid Q$	composition
$!P$	replication
$\mathbf{0}$	inactivity

As in the typed calculus, the names $y_1, \ldots, y_n$ are bound in an input $x(y_1, \ldots, y_n).P$ with scope P, and the name x is bound in $(\nu x)P$ with scope P. We identify processes up to capture-avoiding renaming of bound names. We let $fn(P)$ be the set of names free in P.

Every typed process has a corresponding untyped process obtained by erasing type annotations and group creation operators. We confer a reduction semantics on untyped processes that corresponds to the reduction semantics for typed processes. To describe the possible external interactions of a process we recall standard definitions of labelled input and output transitions.

Reduction: $P \rightarrow Q$

$\overline{x}\langle y_1, \ldots, y_n\rangle \mid x(z_1, \ldots, z_n).P \rightarrow P\{z_1 \leftarrow y_1\} \cdots \{z_n \leftarrow y_n\}$	(Red I/O)
$P \rightarrow Q \Rightarrow P \mid R \rightarrow Q \mid R$	(Red Par)
$P \rightarrow Q \Rightarrow (\nu x)P \rightarrow (\nu x)Q$	(Red Res)
$P' \equiv P, P \rightarrow Q, Q \equiv Q' \Rightarrow P' \rightarrow Q'$	(Red $\equiv$)

Structural Congruence: $P \equiv Q$

$P \equiv P$	(Struct Refl)
$Q \equiv P \Rightarrow P \equiv Q$	(Struct Symm)
$P \equiv Q, Q \equiv R \Rightarrow P \equiv R$	(Struct Trans)
$P \equiv Q \Rightarrow (\nu x)P \equiv (\nu x)Q$	(Struct Res)
$P \equiv Q \Rightarrow P \mid R \equiv Q \mid R$	(Struct Par)
$P \equiv Q \Rightarrow !P \equiv !Q$	(Struct Repl)
$P \equiv Q \Rightarrow x(y_1, \ldots, y_n).P \equiv x(y_1, \ldots, y_n).Q$	(Struct Input)
$P \mid \mathbf{0} \equiv P$	(Struct Par Zero)
$P \mid Q \equiv Q \mid P$	(Struct Par Comm)
$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$	(Struct Par Assoc)
$!P \equiv P \mid !P$	(Struct Repl Par)
$(\nu x)(\nu y)P \equiv (\nu y)(\nu x)P$	(Struct Res Res)
$x \notin fn(P) \Rightarrow (\nu x)(P \mid Q) \equiv P \mid (\nu x)Q$	(Struct Res Par)

From any typed process, we obtain an untyped π-calculus process by erasing type annotations and group restrictions.

Erasures of type annotations and group restrictions:

$$\begin{array}{ll}
\mathit{erase}((\nu G)P) \triangleq \mathit{erase}(P) & \mathit{erase}((\nu x{:}T)P) \triangleq (\nu x)\mathit{erase}(P) \\
\mathit{erase}(\mathbf{0}) \triangleq \mathbf{0} & \mathit{erase}(P \mid Q) \triangleq \mathit{erase}(P) \mid \mathit{erase}(Q) \\
\mathit{erase}(!P) \triangleq !\mathit{erase}(P) & \mathit{erase}(\overline{x}\langle y_1, \dots, y_n\rangle) \triangleq \overline{x}\langle y_1, \dots, y_n\rangle \\
\multicolumn{2}{l}{\mathit{erase}(x(y_1{:}T_1, \dots, y_n{:}T_n).P) \triangleq x(y_1, \dots, y_n).\mathit{erase}(P)}
\end{array}$$

The following proposition shows that although type annotations and group restrictions affect type-checking, they do not affect the dynamic behaviour of a process. We omit the proof; it proceeds along the same lines as a similar result for a related calculus [8].

Proposition 4 (Erasure) *For all typed processes P and Q, $P \rightarrow Q$ implies $\mathit{erase}(P) \rightarrow \mathit{erase}(Q)$ and $\mathit{erase}(P) \rightarrow R$ implies there is a typed process Q such that $P \rightarrow Q$ and $R \equiv \mathit{erase}(Q)$.*

Finally, we define input and output transitions to describe the interactions between an untyped process and an untyped opponent running alongside in parallel. An input transition $P \xrightarrow{x} (y_1, \dots, y_n)Q$ means that P is ready to receive an input tuple on the channel x in the variables $y_1, \dots, y_n$, and then continue as Q. (The variables $y_1, \dots, y_n$ are bound with scope Q.) An output transition $P \xrightarrow{\overline{x}} (\nu z_1, \dots, z_m)\langle y_1, \dots, y_n\rangle Q$ means that P is ready to transmit an output tuple $\langle y_1, \dots, y_n\rangle$ on the channel x, and then continue as Q. The set $\{z_1, \dots, z_m\} \subseteq \{y_1, \dots, y_n\}$ consists of freshly generated names whose scope includes both the tuple $\langle y_1, \dots, y_n\rangle$ and the process Q. The names $z_1, \dots, z_m$ are unknown to the opponent beforehand, but are revealed by the interaction.

Labelled transitions such as these are most commonly defined inductively by a structural operational semantics; for the sake of brevity, the following definitions are in terms of structural congruence.

- Let $P \xrightarrow{x} (y_1, \dots, y_n)Q$ if and only if the names $y_1, \dots, y_n$ are pairwise distinct, and there are processes P_1 and P_2 and pairwise distinct names $z_1, \dots, z_m$ such that $P \equiv (\nu z_1, \dots, z_m)(x(y_1, \dots, y_n).P_1 \mid P_2)$ and $Q \equiv (\nu z_1, \dots, z_m)(P_1 \mid P_2)$ where $x \notin \{z_1, \dots, z_m\}$, and $\{y_1, \dots, y_n\} \cap (\{z_1, \dots, z_m\} \cup \mathit{fn}(P_2)) = \varnothing$.
- Let $P \xrightarrow{\overline{x}} (\nu z_1, \dots, z_m)\langle y_1, \dots, y_n\rangle Q$ if and only if the names $z_1, \dots, z_m$ are pairwise distinct, and we have $P \equiv (\nu z_1, \dots, z_m)(\overline{x}\langle y_1, \dots, y_n\rangle \mid Q)$ where $x \notin \{z_1, \dots, z_m\}$ and $\{z_1, \dots, z_m\} \subseteq \{y_1, \dots, y_n\}$.

We define a (strong, synchronous) bisimilarity on processes as usual [14]: let $\sim$ be the largest symmetric relation such that $P \sim Q$ implies:

(1) whenever $P \rightarrow P'$ there is Q' with $Q \rightarrow Q'$ and $P' \sim Q'$;
(2) whenever $P \xrightarrow{x} (y_1, \ldots, y_n)P'$ there is Q' with $Q \xrightarrow{x} (y_1, \ldots, y_n)Q'$ and for all $z_1, \ldots, z_n$, $P'\{y_1 \leftarrow z_1\} \cdots \{y_n \leftarrow z_n\} \sim Q'\{y_1 \leftarrow z_1\} \cdots \{y_n \leftarrow z_n\}$;
(3) whenever $P \xrightarrow{\overline{x}} (\nu z_1, \ldots, z_m)\langle y_1, \ldots, y_n\rangle P'$ there is Q' with $Q \xrightarrow{\overline{x}} (\nu z_1, \ldots, z_m)\langle y_1, \ldots, y_n\rangle Q'$ and $P' \sim Q'$.

By standard arguments, bisimilarity is a congruence.

3.2 A Secrecy Theorem

The following definition is inspired by Abadi's definition of secrecy [2] in the untyped spi calculus [4]. Abadi attributes the underlying idea to Dolev and Yao [10]: that a name is kept secret from an opponent if after no series of interactions is the name transmitted to the opponent. (In the presence of encryption, the definition is rather more subtle than this.) An alternative we do not pursue here is to formulate secrecy using testing equivalence [1,4].

We model the external opponent simply by the finite set of names S known to it. The four rules displayed below define a relation $(P, S) \ \mathbb{R}_X \ (P', S')$ to mean that starting from a process P and an opponent knowing S, we may reach a state in which P has evolved into P', with fresh names disjoint from the finite set X, and with the opponent now knowing S'. The *frame* X represents the initial knowledge of the process; it is an upper bound on the set of names that are not fresh, and always includes $\mathit{fn}(P)$ (see Lemma 30).We abbreviate the common case $X = \mathit{fn}(P)$ by defining $(P, S) \ \mathbb{R} \ (P', S')$ to mean $(P, S) \ \mathbb{R}_{\mathit{fn}(P)}$ (P', S').

(1) If $\mathit{fn}(P) \subseteq X$ then $(P, S) \ \mathbb{R}_X \ (P, S)$.
(2) If $(P, S) \ \mathbb{R}_X \ (P', S')$ and $P' \rightarrow P''$ then $(P, S) \ \mathbb{R}_X \ (P'', S')$.
(3) If $(P, S) \ \mathbb{R}_X \ (P', S')$, $P' \xrightarrow{x} (y_1, \ldots, y_n)\hat{P}$, $x \in S'$, and $(\{z_1, \ldots, z_n\} - S') \cap X = \varnothing$ then $(P, S) \ \mathbb{R}_X \ (\hat{P}\{y_1 \leftarrow z_1\} \cdots \{y_n \leftarrow z_n\}, S' \cup \{z_1, \ldots, z_n\})$.
(4) If $(P, S) \ \mathbb{R}_X \ (P', S')$, $P' \xrightarrow{\overline{x}} (\nu z_1, \ldots, z_m)\langle y_1, \ldots, y_n\rangle \hat{P}$ and $x \in S'$ and $\{z_1, \ldots, z_m\} \cap (S' \cup X) = \varnothing$ then $(P, S) \ \mathbb{R}_X \ (\hat{P}, S' \cup \{y_1, \ldots, y_n\})$.

Clause (1) says that (P, S) is reachable from itself.

Clause (2) allows the process component to evolve on its own.

Clause (3) allows the process to input the tuple $\langle z_1, \ldots, z_n\rangle$ from the opponent, provided the channel x is known to the opponent. The names $\{z_1, \ldots, z_n\} - S'$ are freshly created by the opponent; the condition $(\{z_1, \ldots, z_n\} - S') \cap X = \varnothing$

ensures these fresh names are not confused with names in the frame X.

Clause (4) allows the process to output the tuple $\langle y_1, \dots, y_n \rangle$ to the opponent, who then knows the names $S \cup \{y_1, \dots, y_n\}$, provided the channel x is known to the opponent. The names $\{z_1, \dots, z_m\}$ (included in $\{y_1, \dots, y_n\}$) are freshly created by the process; the condition $\{z_1, \dots, z_m\} \cap (S' \cup X) = \varnothing$ ensures these fresh names are not confused with names currently known by the opponent or in the frame X.

We now define when a process may reveal a name to an opponent, and its logical negation, when a process preserves the secrecy of a name:

- P *may reveal* x to S if and only if there exist P' and S' such that $(P, S) \mathbb{R} (P', S')$ and $x \in \mathit{fn}(P) \cap (S' - S)$;
- P *preserves the secrecy* of x from S if and only if for all P' and S', $(P, S) \mathbb{R} (P', S')$ implies that $x \notin \mathit{fn}(P) \cap (S' - S)$.

By definition, if P may reveal x to S then $x \in \mathit{fn}(P)$ and $x \notin S$. Conversely, if either $x \notin \mathit{fn}(P)$ or $x \in S$ then P preserves the secrecy of x from S. Intuitively, a process cannot reveal a name if the opponent already knows it, or if the process does not itself know it.

A preliminary version version of this paper [6] formulates name revelation using the condition $x \in \mathit{fn}(S')$ instead of the condition $x \in \mathit{fn}(P) \cap (S' - S)$ above. According to this variation, a process P may reveal x to S even though $x \notin \mathit{fn}(P)$; for example, $P = y(z).\mathbf{0}$ may reveal x to $S = \{y\}$ because we can derive $(y(z).\mathbf{0}, \{y\}) \mathbb{R} (\mathbf{0}, \{x, y\})$, where in effect the opponent is allowed to pick x as a fresh name. Let $Q = (\nu w)\overline{w}\langle x \rangle$. We have that $Q \sim \mathbf{0}$ and therefore that $P \mid Q \sim P$. But $P \mid Q$ does not reveal x to S because $x \in \mathit{fn}(Q)$ and therefore the opponent cannot pick it as a fresh name. This example shows that the definition with the condition $x \in S'$ instead of $x \in \mathit{fn}(P) \cap (S' - S)$ is not preserved by bisimilarity. Preservation by bisimilarity seems a useful guide when formulating secrecy. The intermediate condition $x \in \mathit{fn}(P) \cap S'$, which allows x to be known initially to the opponent, also fails to preserve bisimilarity; we would have that P does not reveal x to $\{x\}$ (because x is not free in P) but $P \mid Q$ does reveal x to $\{x\}$ (because x is free in Q).

The next proposition, proved in Appendix B, is that the definitions displayed above of revelation and secrecy preservation are indeed invariant with respect to bisimilarity.

Proposition 5 *Suppose that $P \sim Q$. If P may reveal x to S then so does Q. Dually, if P preserves the secrecy of x from S then so does Q.*

Our main technical result formalizes the secrecy property of group creation discussed in Section 1.

Theorem 1 (Secrecy) *Let S be the names occurring in $dom(E)$. Suppose that $G \in fg(T)$ and $E \vdash (\nu G)(\nu x{:}T)P$, and hence that $x \notin S$. Then the untyped process $erase(P)$ preserves the secrecy of x from S.*

We give the proof in the next section. The group restriction (νG) is essential. A typing $E \vdash (\nu x{:}T)P$ with $G \in fg(T)$ does not in general imply that the erasure $erase(P)$ preserves the secrecy of x. For example, consider the typing $\varnothing, G, y{:}G[G[\,]] \vdash (\nu x{:}G[\,])\overline{y}\langle x \rangle$. Then the erasure $\overline{y}\langle x \rangle$ reveals x to $S = \{y\}$.

Still, inspired by a result of Sangiorgi and Walker, we can rephrase Theorem 1 without group restriction as follows. Sangiorgi and Walker's result [18, Theorem 9.3.1], in a sorted polyadic π-calculus without sort creation, uses sorts to show the secrecy of a restricted name relative to a well-sorted opponent. Our Theorem 1 is stronger, not only in that it deals with group creation, but also in that it establishes secrecy relative to an untyped opponent. By reformulating the premise of Sangiorgi and Walker's result in our π-calculus, we obtain the following corollary of Theorem 1. In fact, Corollary 1 is effectively a restatement of Theorem 1, as each is a corollary of the other.

Corollary 1 *Let S be the names occurring in $dom(E)$. Suppose $G \in fg(T)$ but for every entry $x'{:}T'$ in E, that $G \notin T'$. If $E \vdash (\nu x{:}T)P$ then the untyped process $erase(P)$ preserves the secrecy of x from S.*

Proof From $E \vdash (\nu x{:}T)P$ it follows that $E \vdash T$, and since $G \in fg(T)$, it must be that $E = E', G, E''$. From $E', G, E'' \vdash (\nu x{:}T)P$ and our assumption that G appears in no type listed in E, we can conclude $E', E'', G \vdash (\nu x{:}T)P$ by repeated use of the exchange lemmas, Lemmas 13 and 14 in Appendix A. By (Proc GRes), $E', E'' \vdash (\nu G)(\nu x{:}T)P$. Hence, Theorem 1 implies that $erase(P)$ preserves the secrecy of x from S. □

4 Proof of Secrecy

The proof of the secrecy theorem is based on an auxiliary type system that partitions channels into untrusted channels, with type Un, and trusted ones, with type $Ch[T_1, \ldots, T_n]$, where each T_i is either a trusted or untrusted type. The type system insists that names are bound to variables with the same trust level (that is, the same type), and that no trusted name is ever transmitted on an untrusted channel. Hence an opponent knowing only untrusted channel names will never receive any trusted name.

For any fixed group G, we can translate group-based types into the auxiliary type system as follows: any type that does not contain G free becomes Un, while a type $H[T_1, \ldots, T_n]$ that contains G free is mapped onto $Ch[T_1', \ldots, T_n']$, where the types $T'_1, \ldots, T'_n$ are the translations of the types $T_1, \ldots, T_n$, respectively. This translation is proved to preserve typability. This implies that an opponent knowing only names whose type does not contain G free, will never be able to learn any name whose type contains G free. This is the key step in proving the secrecy theorem.

Types:

$T ::=$	channel type
$\quad Ch[T_1, \ldots, T_n]$	trusted polyadic channel
$\quad Un$	untrusted name

Judgments:

$E \vdash \diamond$	good environment
$E \vdash x : T$	good name x of type T
$E \vdash P$	good process P

Rules:

(Env $\varnothing$)
$$\frac{}{\varnothing \vdash \diamond}$$

(Env x)
$$\frac{E \vdash \diamond \quad x \notin dom(E)}{E, x{:}T \vdash \diamond}$$

(Exp x)
$$\frac{E', x{:}T, E'' \vdash \diamond}{E', x{:}T, E'' \vdash x : T}$$

(Proc Res)
$$\frac{E, x{:}T \vdash P}{E \vdash (\nu x)P}$$

(Proc Zero)
$$\frac{E \vdash \diamond}{E \vdash \mathbf{0}}$$

(Proc Par)
$$\frac{E \vdash P \quad E \vdash Q}{E \vdash P \mid Q}$$

(Proc Repl)
$$\frac{E \vdash P}{E \vdash !P}$$

(Proc *Ch* Input)
$$\frac{E \vdash x : Ch[T_1, \ldots, T_n] \qquad E, y_1{:}T_1, \ldots, y_n{:}T_n \vdash P}{E \vdash x(y_1, \ldots, y_n).P}$$

(Proc *Ch* Output)
$$\frac{E \vdash x : Ch[T_1, \ldots, T_n] \qquad E \vdash y_1 : T_1 \quad \cdots \quad E \vdash y_n : T_n}{E \vdash \overline{x}\langle y_1, \ldots, y_n \rangle}$$

(Proc *Un* Input)
$$\frac{E \vdash x : Un \qquad E, y_1{:}Un, \ldots, y_n{:}Un \vdash P}{E \vdash x(y_1, \ldots, y_n).P}$$

(Proc *Un* Output)

$$\frac{E \vdash x : Un \qquad E \vdash y_1 : Un \quad \cdots \quad E \vdash y_n : Un}{E \vdash \overline{x}\langle y_1, \ldots, y_n \rangle}$$

The auxiliary type system is defined on untyped processes. Any untrusted opponent may be type-checked, as follows. This property makes this system suitable to reason about a system where trusted and untrusted processes co-exist. The proof is by induction on the size of P.

Lemma 6 *For all P, if $fn(P) = \{x_1, \ldots, x_n\}$ then $\varnothing, x_1{:}Un, \ldots, x_n{:}Un \vdash P$.*

The auxiliary type system enjoys subject congruence and subject reduction. The proofs are in Appendix C.

Lemma 7 (Subject Congruence) *If $E \vdash P$ and $P \equiv Q$ then $E \vdash Q$.*

Proposition 8 (Subject Reduction) *If $E \vdash P$ and $P \to Q$ then $E \vdash Q$.*

The following proposition, proved in Appendix D, is the crux of the proof of Theorem 1: an opponent who knows only untrusted names cannot learn any trusted one.

Proposition 9 *Suppose that $\varnothing, y_1{:}Un, \ldots, y_n{:}Un, x{:}T \vdash P$ where $T \neq Un$. Then the process P preserves the secrecy of the name x from $\{y_1, \ldots, y_n\}$.*

Next, we translate the types and environments of the π-calculus with groups into our auxiliary system, and state that erasure preserves typing.

Translations of types and environments:

$$[\![H[T_1, \ldots, T_n]]\!]_G \triangleq \begin{cases} Ch[[\![T_1]\!]_G, \ldots, [\![T_n]\!]_G] & \text{if } G \in fg(H[T_1, \ldots, T_n]) \\ Un & \text{otherwise} \end{cases}$$

$[\![\varnothing]\!]_G \triangleq \varnothing$
$[\![E, H]\!]_G \triangleq [\![E]\!]_G$
$[\![E, x{:}T]\!]_G \triangleq [\![E]\!]_G, x{:}[\![T]\!]_G$

Appendix D contains a proof of the following:

Proposition 10 *If $E \vdash P$ then $[\![E]\!]_G \vdash erase(P)$.*

We can now prove Theorem 1. We need the following lemma, whose proof is by a routine induction on the derivation of $E, x{:}T, E' \vdash P$.

Lemma 11 *If $E, x{:}T, E' \vdash P$ and $E \vdash y : T$ then $E, E' \vdash P\{x \leftarrow y\}$.*

Restatement of Theorem 1 *Let S be the names occurring in $dom(E)$. Suppose that $G \in fg(T)$ and $E \vdash (\nu G)(\nu x{:}T)P$, and hence that $x \notin S$. Then the untyped process $erase(P)$ preserves the secrecy of x from S.*

Proof Since $E \vdash (\nu G)(\nu x{:}T)P$ must have been derived using (Proc GRes) and (Proc Res), we have $E, G, x{:}T \vdash P$, with $G \notin dom(E)$. Hence, $[\![E]\!]_G = \varnothing, z_1{:}Un, \ldots, z_n{:}Un$ where $S = \{z_1, \ldots, z_n\}$. Proposition 10 implies that $\varnothing, z_1{:}Un, \ldots, z_n{:}Un, x{:}[\![T]\!]_G \vdash erase(P)$. Since $G \in fg(T)$, $[\![T]\!]_G \neq Un$. So Proposition 9 implies that $erase(P)$ preserves the secrecy of x from S. $\square$

5 Conclusion

We proposed a typed π-calculus in which each name belongs to a group, and in which groups may be created dynamically by a group creation operator. Typing rules bound the communication of names of dynamically created groups, hence preventing the accidental or malicious revelation of secrets. We explained these ideas informally, proposed a formalization based on Abadi's notion of name secrecy, and explained the ideas underlying the proof.

The idea of name groups and a group creation operator arose in our recent work on type systems for regulating mobile computation in the ambient calculus [7]. The new contributions of the present paper are to recast the idea in the simple setting of the π-calculus and to explain, formalize, and prove the secrecy properties induced by group creation. The typed π-calculus of Section 2 is extended with an effect system to establish a formal connection between group creation and the *letregion* construct of Tofte and Talpin's region-based memory management [20] in another paper [8]. That other paper generalizes our subject congruence, subject reduction, and erasure results (Lemma 1, Propositions 2 and 4) to the system of types and effects for the π-calculus. We conjecture that the main secrecy result of this paper, Theorem 1, would also hold for that extended system, but we have not studied the details.

The idea of proving a secrecy property for a type system by translation into a mixed trusted and untrusted type system appears to be new. Our work develops the idea of a type system for the π-calculus that mixes trusted and untrusted data, and the idea that every opponent should be typable in the sense of Lemma 6. These ideas first arose in Abadi's type system for the spi calculus [1]. In that system, each name belongs to a global security level, such as *Public* or *Secret*, but there is no level creation construct akin to group creation.

A related paper [5] presents a control flow analysis for the π-calculus that can also establish secrecy properties of names. There is an intriguing connection, that deserves further study, between the groups of our system, and the channels and binders of the flow analysis. One difference between the studies is that the flow analysis has no counterpart of the construct for group creation of this paper. Another is that an algorithm is known for computing flow analyses for the π-calculus, whereas we have not investigated algorithmic aspects of our type system. It would be interesting to consider whether algorithms for Milner's systems of sorts [12,21] extend to our calculus.

A recent paper [3] presents a type system for establishing secrecy properties in a relative of the spi calculus without groups but equipped with primitives for public key cryptography. It presents a useful new definition of name secrecy based on a reduction relation rather than the labelled transitions of this paper.

Other related work on the π-calculus includes type systems for guaranteeing locality properties [17,19]. These systems can type-check whether a name may leak outside a particular locality.

In summary, group creation is a powerful new construct for process calculi. Its study is just beginning; we expect that its secrecy guarantees will help with the design and semantics of new programming language features, and with the analysis of security properties of individual programs.

A Type Preservation for the Main Type System

This appendix contains proofs of subject congruence (Lemma 1) and subject reduction (Proposition 2) for the main type system.

Lemma 12 *If* $E, n{:}T, m{:}T', E' \vdash \mathbb{J}$ *then* $E, m{:}T', n{:}T, E' \vdash \mathbb{J}$.

Lemma 13 *If* $E, G, n{:}T, E' \vdash \mathbb{J}$ *and* $G \notin fn(T)$ *then then* $E, n{:}T, G, E' \vdash \mathbb{J}$.

Lemma 14 *If* $E, G, G', E' \vdash \mathbb{J}$ *then* $E, G', G, E' \vdash \mathbb{J}$.

Lemma 15 *If* $E, x{:}T, E' \vdash \mathbb{J}$ *and* $x \notin fn(\mathbb{J})$ *then* $E, E' \vdash \mathbb{J}$.

Lemma 16 *If* $E, G, E' \vdash \mathbb{J}$ *and* $G \notin fn(\mathbb{J}) \cup fn(E')$ *then* $E, E' \vdash \mathbb{J}$.

Lemma 17 *If* $E, E' \vdash \mathbb{J}$ *and* $E, x : T, E' \vdash \mathbb{J}'$ *then* $E, x : T, E' \vdash \mathbb{J}$.

Lemma 18 *If* $E, E' \vdash \mathbb{J}$ *and* $E, G, E' \vdash \mathbb{J}'$ *then* $E, G, E' \vdash \mathbb{J}$.

Lemma 19 $E, E' \vdash \mathbb{J}$ *then* $E \vdash \diamond$.

Lemma 20 *If $E, x{:}T, E' \vdash \mathbb{J}$ and $E \vdash y : T$ then $E, E' \vdash \mathbb{J}\{x{\leftarrow}y\}$.*

Lemma 21 *If $E \vdash x : T$ and $E \vdash x : T'$ then $T = T'$.*

Restatement of Lemma 1 *If $E \vdash P$ and $P \equiv Q$ then $E \vdash Q$.*

Proof The lemma follows by showing that $P \equiv Q$ implies:

(1) If $E \vdash P$ then $E \vdash Q$.
(2) If $E \vdash Q$ then $E \vdash P$.

We proceed by induction on the derivation of $P \equiv Q$.

(Struct Refl) Trivial.
(Struct Symm) Then $Q \equiv P$. For (1), assume $E \vdash P$. By induction hypothesis (2), $Q \equiv P$ implies that $E \vdash Q$. Part (2) is symmetric.
(Struct Trans) Then $P \equiv R$, $R \equiv Q$ for some R. For (1), assume $E \vdash P$. By induction hypothesis (1), $E \vdash R$. Again by induction hypothesis (1), $E \vdash Q$. Part (2) is symmetric.
(Struct Res) Then $P = (\nu x{:}T)P'$ and $Q = (\nu x{:}T)Q'$, with $P' \equiv Q'$. For (1), assume $E \vdash P$. This must have been derived from (Proc Res), with $E, x{:}T \vdash P'$. By induction hypothesis (1), $E, x{:}T \vdash Q'$. By (Proc Res), $E \vdash (\nu x{:}T)Q'$. Part (2) is symmetric.
(Struct GRes) Then $P = (\nu G)P'$ and $Q = (\nu G)Q'$, with $P' \equiv Q'$. For (1), assume $E \vdash P$. This must have been derived from (Proc GRes), with $E, G \vdash P'$. By induction hypothesis (1), $E, G \vdash Q'$. By (Proc GRes), $E \vdash (\nu G)Q'$. Part (2) is symmetric.
(Struct Par) Then $P = P' \mid R$, $Q = Q' \mid R$, and $P' \equiv Q'$. For (1), assume $E \vdash P' \mid R$. This must have been derived from (Proc Par), with $E \vdash P'$, $E \vdash R$. By induction hypothesis (1), $E \vdash Q'$. By (Proc Par), $E \vdash Q' \mid R$. Part (2) is symmetric.
(Struct Repl) Then $P = !P'$, $Q = !Q'$, and $P' \equiv Q'$. For (1), assume $E \vdash P$. This must have been derived from (Proc Repl), with $E \vdash P'$. By induction hypothesis (1), $E \vdash Q'$. By (Proc Repl), $E \vdash !Q'$. Part (2) is symmetric.
(Struct Input) In this case, we have $P = x(y_1{:}T_1, \ldots, y_k{:}T_k).P'$, $Q = x(y_1{:}T_1, \ldots, y_k{:}T_k).Q'$, and $P' \equiv Q'$. For (1), assume $E \vdash P$. This must have been derived from (Proc Input), with $E, y_1{:}T_1, \ldots, y_k{:}T_k \vdash P'$, $E \vdash x : G[T_1, \ldots, T_n]$, for some G. By induction hypothesis, $E, y_1{:}T_1, \ldots, y_k{:}T_k \vdash Q'$. By (Proc Input), $E \vdash x(y_1{:}T_1, \ldots, y_k{:}T_k).Q'$. Part (2) is symmetric.
(Struct Par Zero) Then $P = P' \mid \mathbf{0}$ and $Q = P'$.

For (1), assume $E \vdash P' \mid \mathbf{0}$. This must have been derived from (Proc Par) with $E \vdash P'$.

For (2), assume $E \vdash P'$. By Lemma 19, $E \vdash \diamond$. By (Proc Zero), $E \vdash \mathbf{0}$ By (Proc Par), $E \vdash P' \mid \mathbf{0}$

(Struct Par Comm) Then $P = P' \mid P''$ and $Q = P'' \mid P'$.

For (1), assume $E \vdash P' \mid P''$. This must have been derived from $E \vdash P'$

and $E \vdash P''$. By (Proc Par), $E \vdash P'' \mid P'$. Hence, $E \vdash Q$.

Part (2) is symmetric.

(Struct Par Assoc) Then $P = (P' \mid P'') \mid P'''$ and $Q = P' \mid (P'' \mid P''')$.

For (1), assume $E \vdash (P' \mid P'') \mid P'''$. This must have been derived from (Proc Par) twice, with $E \vdash P'$, $E \vdash P''$, and $E \vdash P'''$. By (Proc Par) twice, $E \vdash P' \mid (P'' \mid P''')$. Hence $E \vdash Q$.

Part (2) is symmetric.

(Struct Repl Par) Then $P = !P'$ and $Q = P' \mid !P'$. For (1), assume $E \vdash !P'$. This must have been derived from (Proc Repl), with $E \vdash P'$. By (Proc Par), $E \vdash P' \mid !P'$. Hence, $E \vdash Q$.

For (2), assume $E \vdash P' \mid !P'$. This must have been derived from (Proc Par), with $E \vdash P'$ and $E \vdash !P'$. Hence, $E \vdash P$.

(Struct Res Res) In this case we have $P = (\nu x_1{:}T_1)(\nu x_2{:}T_2)P'$ and $Q = (\nu x_2{:}T_2)(\nu x_1{:}T_1)P'$ with $x_1 \neq x_2$.

For (1), assume $E \vdash (\nu x_1{:}T_1)(\nu x_2{:}T_2)P'$. This must have been derived from (Proc Res) twice, with $E, x_1{:}T_1, x_2{:}T_2 \vdash P'$. By Lemma 12, we have $E, x_2{:}T_2, x_1{:}T_1 \vdash P'$. By (Proc Res) twice we have $E \vdash (\nu x_2{:}T_2)(\nu x_1{:}T_1)P'$. Part (2) is symmetric.

(Struct Res Par) Then $P = (\nu x{:}T)(P' \mid P'')$ and $Q = P' \mid (\nu x{:}T)P''$, with $x \notin fn(P')$.

For (1), assume $E \vdash P$. This must have been derived from (Proc Res), with $E, x{:}T \vdash P' \mid P''$. and from (Proc Par), with $E, x{:}T \vdash P'$ and $E, x{:}T \vdash P''$. By Lemma 15, since $x \notin fn(P')$, we have $E \vdash P'$. By (Proc Res) we have $E \vdash (\nu x{:}T)P''$. By (Proc Par) we have $E \vdash P' \mid (\nu x{:}T)P''$, that is, $E \vdash Q$.

For (2), assume $E \vdash Q$. This must have been derived from (Proc Par), with $E \vdash P'$ and $E \vdash (\nu x{:}T)P''$, and from (Proc Res), with $E, x{:}T \vdash P''$. By Lemma 17, $E, x{:}T \vdash P'$. By (Proc Par), $E, x{:}T \vdash P' \mid P''$. By (Proc Res), $E \vdash (\nu x{:}T)(P' \mid P'')$, that is, $E \vdash P$.

(Struct GRes GRes) Then $P = (\nu G_1)(\nu G_2)P'$ and $Q = (\nu G_2)(\nu G_1)P'$.

For (1), assume $E \vdash (\nu G_1)(\nu G_2)P'$. This must have been derived from (Proc GRes) twice, with $E, G_1, G_2 \vdash P'$. By Lemma 14 we have $E, G_2, G_1 \vdash P'$. By (Proc Res) twice we have $E \vdash (\nu G_2)(\nu G_1)P'$.

Part (2) is symmetric.

(Struct GRes Res) Then $P = (\nu G)(\nu x{:}T)P'$ and $Q = (\nu x{:}T)(\nu G)P'$ with $G \notin fg(T)$.

For (1), assume $E \vdash (\nu G)(\nu x{:}T)P'$. This must have been derived from (Proc GRes), with $E, G \vdash (\nu x{:}T)P'$, and from (Proc Res), with $E, G, x{:}T \vdash P'$. Since $G \notin fg(T)$ by hypothesis, by Lemma 13 we have $E, x{:}T, G \vdash P'$. Then, by (Proc GRes) and (Proc Res) we have $E \vdash (\nu x{:}T)(\nu G)P'$.

Part (2) is symmetric.

(Struct GRes Par) Then $P = (\nu G)(P' \mid P'')$ and $Q = P' \mid (\nu G)P''$, with $G \notin fg(P')$.

For (1), assume $E \vdash P$. This must have been derived from (Proc GRes), with $E, G \vdash P' \mid P''$, and from (Proc Par), with $E, G \vdash P'$ and $E, G \vdash P''$.

By Lemma 16, since $G \notin fg(P')$, we have $E \vdash P'$. By (Proc GRes) we have $E \vdash (\nu G)P''$. By (Proc Par) we have $E \vdash P' \mid (\nu G)P''$, that is, $E \vdash Q$.

For (2), assume $E \vdash Q$. This must have been derived from (Proc Par), with $E \vdash P'$ and $E \vdash (\nu G)P''$, and from (Proc GRes), with $E, G \vdash P''$. By Lemma 18, $E, G \vdash P'$. By (Proc Par), $E, G \vdash P' \mid P''$. By (Proc GRes), $E \vdash (\nu G)(P' \mid P'')$, that is, $E \vdash P$. $\square$

Restatement of Proposition 2 *If $E \vdash P$ and $P \to Q$ then $E \vdash Q$.*

Proof By induction on the derivation of $P \to Q$.

(Red I/O) Then $P = \overline{x}\langle y_1, \ldots, y_k\rangle \mid x(z_1{:}T_1, \ldots, z_k{:}T_k).P'$ and $Q = P'\{z_1 \leftarrow y_1\} \cdots \{z_k \leftarrow y_k\}$. Assume $E \vdash P$. This must have been derived from (Proc Par) with $E \vdash x(z_1{:}T_1, \ldots, z_k{:}T_k).P'$ and $E \vdash \overline{x}\langle y_1, \ldots, y_k\rangle$. The former must have been derived from (Proc Input) with $E \vdash x : G[T_1, \ldots, T_k]$, $E, z_1{:}T_1, \ldots, z_k{:}T_k \vdash P'$. The latter judgment $E \vdash \overline{x}\langle y_1, \ldots, y_k\rangle$ must have been derived from (Proc Output) with $E \vdash x : G[T'_1, \ldots, T'_k]$ $E \vdash y_i : T'_i$ for each $i \in 1..k$. By Lemma 21, $T_i = T'_i$ for each $i \in 1..k$. By k applications of Lemma 20, we get $E \vdash P'\{z_1 \leftarrow y_1\} \cdots \{z_k \leftarrow y_k\}$.

(Red Par) Here $P = P' \mid R$ and $Q = Q' \mid R$ with $P' \to Q'$. Assume $E \vdash P$. This must have been derived using (Proc Par) from $E \vdash P'$ and $E \vdash R$. By induction hypothesis, $E \vdash Q'$. By (Proc Par), $E \vdash Q' \mid R$, that is, $E \vdash Q$.

(Red GRes) Here $P = (\nu G)P'$ and $Q = (\nu G)Q'$ with $P' \to Q'$. Assume $E \vdash P$. This must have been derived using (Proc GRes) from $E, G \vdash P'$. By induction hypothesis, $E, G \vdash Q'$. By (Proc GRes), $E \vdash (\nu G)Q'$, that is, $E \vdash Q$.

(Red Res) Here $P = (\nu x{:}T)P'$ and $Q = (\nu x{:}T)Q'$ with $P' \to Q'$. Assume $E \vdash P$. This must have been derived using (Proc Res) from $E, x{:}T \vdash P'$. By induction hypothesis, $E, x{:}T \vdash Q'$. By (Proc Res), $E \vdash (\nu x{:}T)Q'$, that is, $E \vdash Q$.

(Red $\equiv$) Here $P \equiv P'$, $P' \to Q'$, and $Q' \equiv Q$. Assume $E \vdash P$. By Lemma 1, $E \vdash P'$. By induction hypothesis, $E \vdash Q'$. By Lemma 1, $E \vdash Q$. $\square$

B Facts Needed in Proof That Bisimilarity Preserves Secrecy

This appendix contains a proof of Proposition 5, together with auxiliary lemmas. For the sake of brevity, in this appendix we often use vector notations for name sequences, such as $\vec{y}$ for $y_1, \ldots, y_n$.

We assert three basic lemmas, which may be proved by routine inductions on the length of inference of $(P, S) \mathbb{R}_X (P', S')$. In the first, we can interpret the equation $S' = S \uplus S_P \uplus S_N$ as meaning that in the final state, the knowledge S' of the opponent consists of S, the names it knew to begin with, plus S_P, names known only to P initially, but now revealed, plus S_N, fresh names generated either by P or the opponent. The second and third lemmas state that in some circumstances adding or removing a name y to the frame does not affect reachability.

Lemma 22 *If $(P,S)\ \mathbb{R}_X\ (P',S')$ then (1) $fn(P) \subseteq X$ and (2) $fn(P') \subseteq fn(P) \cup S'$ and (3) $S' = S \uplus S_P \uplus S_N$ with $S_P \subseteq fn(P)$ and $S_N \cap X = \varnothing$.*

Lemma 23 *If $(P,S)\ \mathbb{R}_X\ (P',S')$ and either $y \notin S'$ or $y \in S$ then $(P,S)\ \mathbb{R}_{X\cup\{y\}}\ (P',S')$.*

Lemma 24 *If $(P,S)\ \mathbb{R}_{X\cup\{y\}}\ (P',S')$ and $y \notin fn(P)$ then $(P,S)\ \mathbb{R}_X\ (P',S')$.*

Next, we give a lemma saying that a freshly generated name y can always be renamed to a fresh name z.

Lemma 25 *If $(P,S)\ \mathbb{R}_X\ (P',S' \uplus \{y\})$ and $y \notin S \cup X$ and $z \notin X \cup S'$ then $(P,S)\ \mathbb{R}_X\ (P'\{y\leftarrow z\}, S' \uplus \{z\})$.*

Proof By induction on the length of inference of $(P,S)\ \mathbb{R}_X\ (P',S' \uplus \{y\})$. By hypothesis, $y \notin S \cup X$ and $z \notin X \cup S'$, and we may assume $z \neq y$ (or else the claim is trivial) and consider four cases:

(1) We have $(P,S)\ \mathbb{R}_X\ (P',S' \uplus \{y\})$ from $P = P'$, $S = S' \uplus \{y\}$, and $fn(P) \subseteq X$. But this contradicts the assumption that $y \notin S \cup X$.

(2) We have $(P,S)\ \mathbb{R}_X\ (P',S' \uplus \{y\})$ from $(P,S)\ \mathbb{R}_X\ (P'',S' \uplus \{y\})$ and $P'' \to P'$. By induction hypothesis, $(P,S)\ \mathbb{R}_X\ (P''\{y\leftarrow z\}, S' \uplus \{z\})$. Substitution preserves reduction, so we have $P''\{y\leftarrow z\} \to P'\{y\leftarrow z\}$. Hence, we derive $(P,S)\ \mathbb{R}_X\ (P'\{y\leftarrow z\}, S' \uplus \{z\})$.

(3) We have $(P,S)\ \mathbb{R}_X\ (Q\{\vec{y}\leftarrow\vec{z}\}, S'' \cup \{\vec{z}\})$ from $P'' \xrightarrow{x} (\vec{y})Q$, $x \in S''$, and $(\{\vec{z}\} - S'') \cap X = \varnothing$, and $(P,S)\ \mathbb{R}_X\ (P'',S'')$, where $y \in S'' \cup \{\vec{z}\}$ but $z \notin X \cup S'' \cup \{\vec{z}\}$. We may assume that the bound variables $\vec{y}$ do not include y or z.

Either $y \in S''$ or not. If so, we have $S'' = S''' \uplus \{y\}$ and $S''' \subseteq S'$. By induction hypothesis, $(P,S)\ \mathbb{R}_X\ (P''\{y\leftarrow z\}, S''' \uplus \{z\})$. Substitution preserves transitions, so $P''\{y\leftarrow z\} \xrightarrow{x\{y\leftarrow z\}} (\vec{y})(Q\{y\leftarrow z\})$. We calculate:

- $x\{y\leftarrow z\} \in S''' \uplus \{z\}$ (because $x \in S'' = S''' \uplus \{y\}$)
- $(\{\vec{z}\{y\leftarrow z\}\} - S''\{y\leftarrow z\}) \cap X = \varnothing$ (because $(\{\vec{z}\} - S'') \cap X = \varnothing$ and $z \notin X$)
- $Q\{y\leftarrow z\}\{\vec{y}\leftarrow\vec{z}\{y\leftarrow z\}\} = Q\{\vec{y}\leftarrow\vec{z}\}\{y\leftarrow z\}$
- $(S''' \uplus \{z\}) \cup \vec{z}\{y\leftarrow z\} = (S'' \cup \{\vec{z}\})\{y\leftarrow z\})$

Therefore, $(P,S)\ \mathbb{R}_X\ (Q\{\vec{y}\leftarrow\vec{z}\}\{y\leftarrow z\}, (S'' \cup \{\vec{z}\})\{y\leftarrow z\})$, as required.

On the other hand, suppose that $y \notin S''$, and therefore that $y \in \{\vec{z}\}$, but $y \notin fn(Q)$ (by Lemma 22, parts (1) and (2)). We have $(\{\vec{z}\{y\leftarrow z\}\} - S'') \cap X = \varnothing$ (because $(\{\vec{z}\} - S'') \cap X = \varnothing$ and $z \notin X$). Hence, we can derive $(P,S)\ \mathbb{R}_X\ (Q\{\vec{y}{\leftarrow}\vec{z}\{y{\leftarrow}z\}\}, S''\cup\{\vec{z}\{y{\leftarrow}z\}\})$. We calculate:

- $Q\{\vec{y}{\leftarrow}\vec{z}\{y{\leftarrow}z\}\} = Q\{\vec{y}{\leftarrow}\vec{z}\}\{y{\leftarrow}z\}$ (because $y\notin \mathit{fn}(Q)$)
- $S''\cup\{\vec{z}\{y{\leftarrow}z\}\} = (S''\cup\{\vec{z}\})\{y{\leftarrow}z\})$ (because $y\notin S''$)

Again, we may conclude $(P,S)\ \mathbb{R}_X\ (Q\{\vec{y}{\leftarrow}\vec{z}\}\{y{\leftarrow}z\}, (S''\cup\{\vec{z}\})\{y{\leftarrow}z\})$.

(4) We have $(P,S)\ \mathbb{R}_X\ (Q, S''\cup\{\vec{y}\})$ from $(P,S)\ \mathbb{R}_X\ (P'',S'')$, $P''\xrightarrow{\overline{x}}(\nu\vec{z})\langle\vec{y}\rangle Q$, $x\in S''$, and $\{\vec{z}\}\cap(S''\cup X)=\varnothing$, with $y\in S''\cup\{\vec{y}\}$ but $z\notin S''\cup\{\vec{y}\}$.

Either $y\in S''$ or not. If so, note that neither y nor z is among the variables $\vec{z}$. By induction hypothesis, we have $(P,S)\ \mathbb{R}_X\ (P''\{y{\leftarrow}z\}, S''\{y{\leftarrow}z\})$. Substitution preserves transitions, so $P''\{y{\leftarrow}z\}\xrightarrow{\overline{x\{y\leftarrow z\}}}(\nu\vec{z})\langle\vec{y}\{y{\leftarrow}z\}\rangle(Q\{y{\leftarrow}z\})$. We have $x\{y{\leftarrow}z\}\in S''\{y{\leftarrow}z\}$ (because $x\in S''$). We have $\{\vec{z}\}\cap(S''\{y{\leftarrow}z\}\cup X)=\varnothing$ (because $\{\vec{z}\}\cap(S''\cup X)=\varnothing$ and neither y nor z is among $\vec{z}$). We have $S''\{y{\leftarrow}z\}\cup\{\vec{y}\{y{\leftarrow}z\}\} = (S''\cup\{\vec{y}\})\{y{\leftarrow}z\}$. Hence, we can derive $(P,S)\ \mathbb{R}_X\ (Q\{y{\leftarrow}z\}, (S''\cup\{\vec{y}\})\{y{\leftarrow}z\})$ as desired.

On the other hand, suppose that $y\notin S''$ so that $y\in\{\vec{y}\}$. By Lemma 22, $\mathit{fn}(P'')\subseteq S\cup X$, and by definition $\{\vec{y}\}\subseteq \mathit{fn}(P'')\cup\{\vec{z}\}$. Since $y\notin S\cup X$ it must be that $y\in\{\vec{z}\}$. By fresh renaming, we have $P''\xrightarrow{\overline{x}}(\nu\vec{z}\{y{\leftarrow}z\})\langle\vec{y}\{y{\leftarrow}z\}\rangle(Q\{y{\leftarrow}z\})$. We have $\{\vec{z}\{y{\leftarrow}z\}\}\cap(S''\cup X)=\varnothing$ (because $\{\vec{z}\}\cap(S''\cup X)=\varnothing$ and $z\notin S''\cup X$). We have $S''\cup\{\vec{y}\{y{\leftarrow}z\}\} = (S''\cup\{\vec{y}\})\{y{\leftarrow}z\}$ (because $y\notin S''$). Hence, we can again derive $(P,S)\ \mathbb{R}_X\ (Q\{y{\leftarrow}z\}, S''\cup\{\vec{y}\{y{\leftarrow}z\}\})$. □

Using the previous lemmas, we show that adding any name to the frame preserves reachability in the following sense:

Lemma 26 *Suppose $(P,S)\ \mathbb{R}_X\ (P',S')$. Then for any y there are P'', S'' such that $(P,S)\ \mathbb{R}_{X\cup\{y\}}\ (P'',S'')$ and $X\cap S' = X\cap S''$.*

Proof We can assume $y\notin X$ or else the lemma is trivial. By Lemma 23, if either $y\notin S'$ or $y\in S$ we get $(P,S)\ \mathbb{R}_{X\cup\{y\}}\ (P',S')$, and so we are done. Otherwise, we have $S'=\hat{S}\uplus\{y\}$ for some S'', and $y\notin S\cup X$. By Lemma 25, we pick some $z\notin X\cup S'$, and obtain $(P,S)\ \mathbb{R}_X\ (P'\{y{\leftarrow}z\}, \hat{S}\uplus\{z\})$. We have $y\notin S'\uplus\{z\}$, so Lemma 23, implies $(P,S)\ \mathbb{R}_{X\cup\{y\}}\ (P'\{y{\leftarrow}z\}, \hat{S}\uplus\{z\})$. Finally, since neither $y\in X$ nor $z\in X$, we have $X\cap(\hat{S}\uplus\{y\}) = X\cap(\hat{S}\uplus\{z\})$. □

An intuition for the next lemma is that the set of names revealed by a transition $(P,S)\ \mathbb{R}_X\ (P',S')$, could be defined equally in terms of either $\mathit{fn}(P)$ or X.

Lemma 27 *If $(P,S)\ \mathbb{R}_X\ (P',S')$ then $\mathit{fn}(P)\cap(S'-S) = X\cap(S'-S)$.*

Proof By Lemma 22, $\mathit{fn}(P)\subseteq X$ and there are S_P and S_N such that $S'=S\uplus S_P\uplus S_N$ with $S_P\subseteq \mathit{fn}(P)$ and $S_N\cap X=\varnothing$. Now $S'-S = S_P\uplus S_N$ so $\mathit{fn}(P)\cap(S'-S) = S_P = X\cap(S'-S)$. □

The frame X appearing in the definition of reachability $(P, S)\ \mathbb{R}_X\ (P', S')$ is a finite set of names, including $fn(P)$, that cannot be chosen as fresh names. In our definition of name revelation, we take $X = fn(P)$. The following lemma, in the style of Gabbay and Pitts' result about their freshness quantifier [11], shows that the exact choice of X does not matter so long as $X \supseteq fn(P)$.

Lemma 28 *These are equivalent:*

(1) *P may reveal x to S*
(2) $\exists X \supseteq fn(P).\exists P', S'.(P, S)\ \mathbb{R}_X\ (P', S') \wedge x \in fn(P) \cap (S' - S)$
(3) $\forall X \supseteq fn(P).\exists P', S'.(P, S)\ \mathbb{R}_X\ (P', S') \wedge x \in fn(P) \cap (S' - S)$

Proof We may assume that $x \in fn(P)$ and $x \notin S$ for otherwise (1), (2), and (3) are false. Hence, it suffices to show equivalence of the following:

(R) $\exists P', S'.(P, S)\ \mathbb{R}_{fn(P)}\ (P', S') \wedge x \in S'$
(E) $\exists n, y_1, \ldots, y_n \notin fn(P).\exists P', S'.(P, S)\ \mathbb{R}_{fn(P)\cup\{y_1,\ldots,y_n\}}\ (P', S') \wedge x \in S'$
(A) $\forall n, y_1, \ldots, y_n \notin fn(P).\exists P', S'.(P, S)\ \mathbb{R}_{fn(P)\cup\{y_1,\ldots,y_n\}}\ (P', S') \wedge x \in S'$

We can obtain (E) from (R) by putting $n = 0$, and obtain (R) from (E) by n applications of Lemma 24. Moreover, we can obtain (R) from (A) by putting $n = 0$, and obtain (A) from (R) by n applications of Lemma 26, establishing that $(P, S)\ \mathbb{R}_{fn(P)\cup\{y_1,\ldots,y_n\}}\ (P'', S'')$ and $fn(P) \cap S' = fn(P) \cap S''$, and hence that $x \in S''$. So all three properties are equivalent. □

We note the following, a corollary by negation:

Lemma 29 *These are equivalent:*

(1) *P preserves the secrecy of x from S*
(2) $\forall X \supseteq fn(P).\forall P', S'.(P, S)\ \mathbb{R}_X\ (P', S') \Rightarrow x \notin fn(P) \cap (S' - S)$
(3) $\exists X \supseteq fn(P).\forall P', S'.(P, S)\ \mathbb{R}_X\ (P', S') \Rightarrow x \notin fn(P) \cap (S' - S)$

The heart of the proof of Proposition 5, below, is that bisimilarity preserves reachability, in the following sense:

Lemma 30 *If $(P, S)\ \mathbb{R}_X\ (P', S')$ and $P \sim Q$ and $fn(P) \cup fn(Q) \subseteq X$ then there is Q' such that $(Q, S)\ \mathbb{R}_X\ (Q', S')$ and $P' \sim Q'$.*

Proof By induction on the length of the derivation of $(P, S)\ \mathbb{R}_X\ (P', S')$.

(1) We have $(P, S)\ \mathbb{R}_X\ (P, S)$ from $fn(P) \subseteq X$. Since $fn(Q) \subseteq X$ we also have $(Q, S)\ \mathbb{R}_X\ (Q, S)$.
(2) We have $(P, S)\ \mathbb{R}_X\ (P'', S')$ from $(P, S)\ \mathbb{R}_X\ (P', S')$ and $P' \to P''$. By induction hypothesis, $(Q, S)\ \mathbb{R}_X\ (Q', S')$ for some Q' with $P' \sim Q'$. Hence, $Q' \to Q''$ for some Q'' with $P'' \sim Q''$. So we can derive $(Q, S)\ \mathbb{R}_X\ (Q'', S')$.
(3) We have $(P, S)\ \mathbb{R}_X\ (\hat{P}\{\vec{y}{\leftarrow}\vec{z}\}, S' \cup \{\vec{z}\})$ from $(P, S)\ \mathbb{R}_X\ (P', S')$, $P' \xrightarrow{x}$

$(\vec{y})\hat{P}$, $x \in S'$, and $(\{\vec{z}\}-S') \cap X = \varnothing$. By induction hypothesis, $(Q,S)\ \mathbb{R}_X$ (Q',S') for some Q' with $P' \sim Q'$. Hence, $Q' \xrightarrow{x} (\vec{y})\hat{Q}$ for some $\hat{Q}$ with $\hat{P}\{\vec{y}\leftarrow\vec{z}\} \sim \hat{Q}\{\vec{y}\leftarrow\vec{z}\}$. So we can derive $(Q,S)\ \mathbb{R}_X\ (\hat{Q}\{\vec{y}\leftarrow\vec{z}\}, S' \cup \{\vec{z}\})$.

(4) We have $(P,S)\ \mathbb{R}_X\ (\hat{P}, S' \cup \{\vec{y}\})$ from $(P,S)\ \mathbb{R}_X\ (P',S')$, $P' \xrightarrow{\overline{x}} (\nu\vec{z})\langle\vec{y}\rangle\hat{P}$ and $x \in S'$ and $\{\vec{z}\} \cap (S' \cup X) = \varnothing$. By induction hypothesis, $(Q,S)\ \mathbb{R}_X\ (Q',S')$ for some Q' with $P' \sim Q'$. Hence, $Q' \xrightarrow{\overline{x}} (\nu\vec{z})\langle\vec{y}\rangle\hat{Q}$ for some $\hat{Q}$ with $\hat{P} \sim \hat{Q}$. So we can derive $(Q,S)\ \mathbb{R}_X\ (\hat{Q}, S' \cup \{\vec{y}\})$. $\square$

Restatement of Proposition 5 *Suppose that $P \sim Q$. If P may reveal x to S then so does Q. Dually, if P preserves the secrecy of x from S then so does Q.*

Proof For the first part, let $X = \mathit{fn}(P) \cup \mathit{fn}(Q)$. By Lemma 28(1,3), there are P' and S' such that $(P,S)\ \mathbb{R}_X\ (P',S')$ and $x \in \mathit{fn}(P) \cap (S' - S)$. By Lemma 27, $x \in X \cap (S' - S)$. By Lemma 30, $P \sim Q$ implies there is Q' such that $(Q,S)\ \mathbb{R}_X\ (Q',S')$ and $P' \sim Q'$. By Lemma 27, $x \in \mathit{fn}(Q) \cap (S' - S)$. By Lemma 28(1,2), Q may reveal x to S.

The second part is a corollary of the first by negation and symmetry. $\square$

C Type Preservation for the Auxiliary Type System

This appendix contains proofs of subject congruence (Lemma 7) and subject reduction (Proposition 8) for the auxiliary type system.

Lemma 31 *If $P \equiv Q$ then $P\{x\leftarrow y\} \equiv Q\{x\leftarrow y\}$.*

Lemma 32 *If $E \vdash \mathbb{J}$ then $E \vdash \diamond$.*

Lemma 33 *If $E', n{:}T', m{:}T'', E'' \vdash \mathbb{J}$ then $E', m{:}T'', n{:}T', E'' \vdash \mathbb{J}$.*

Lemma 34 *If $E', x{:}T, E'' \vdash \mathbb{J}$ and $x \notin \mathit{fn}(\mathbb{J})$ then $E', E'' \vdash \mathbb{J}$.*

Lemma 35 *If $E \vdash \mathbb{J}$ and $E, E' \vdash \diamond$ then $E, E' \vdash \mathbb{J}$.*

Lemma 36 *If $E \vdash x : T$ and $E \vdash x : T'$ then $T = T'$.*

Restatement of Lemma 7 *If $E \vdash P$ and $P \equiv Q$ then $E \vdash Q$.*

Proof The lemma follows by showing that $P \equiv Q$ implies:

(1) If $E \vdash P$ then $E \vdash Q$.
(2) If $E \vdash Q$ then $E \vdash P$.

We proceed by induction on the derivation of $P \equiv Q$.

(Struct Refl) Trivial.

(Struct Symm) Then $Q \equiv P$. For (1), assume $E \vdash P$. By induction hypothesis (2), $Q \equiv P$ implies that $E \vdash Q$. Part (2) is symmetric.

(Struct Trans) Then $P \equiv R$, $R \equiv Q$ for some R. For (1), assume $E \vdash P$. By induction hypothesis (1), $E \vdash R$. Again by induction hypothesis (1), $E \vdash Q$. Part (2) is symmetric.

(Struct Res) Then $P = (\nu x)P'$ and $Q = (\nu x)Q'$, with $P' \equiv Q'$. For (1), assume $E \vdash P$. This must have been derived from (Proc Res), with $E, x{:}T \vdash P'$, for some T. By induction hypothesis (1), $E, x{:}T \vdash Q'$. By (Proc Res), $E \vdash (\nu x)Q'$. Part (2) is symmetric.

(Struct Par) Then $P = P' \mid R$, $Q = Q' \mid R$, and $P' \equiv Q'$. For (1), assume $E \vdash P' \mid R$. This must have been derived from (Proc Par), with $E \vdash P'$, $E \vdash R$. By induction hypothesis (1), $E \vdash Q'$. By (Proc Par), $E \vdash Q' \mid R$. Part (2) is symmetric.

(Struct Repl) Then $P = !P'$, $Q = !Q'$, and $P' \equiv Q'$. For (1), assume $E \vdash P$. This must have been derived from (Proc Repl), with $E \vdash P'$. By induction hypothesis (1), $E \vdash Q'$. By (Proc Repl), $E \vdash !Q'$. Part (2) is symmetric.

(Struct Input) In this case, we have $P = x(y_1, \ldots, y_k).P'$, $Q = x(y_1, \ldots, y_k).Q'$, and $P' \equiv Q'$. For (1), assume $E \vdash P$. This must have been derived either from (Proc *Ch* Input) or from (Proc *Un* Input). In the (Proc *Ch* Input) case, we have that $E \vdash x : Ch[T_1, \ldots, T_n]$, for some $T_1, \ldots, T_n$, and $E, y_1{:}T_1, \ldots, y_k{:}T_k \vdash P'$. By induction hypothesis, $E, y_1{:}T_1, \ldots, y_k{:}T_k \vdash Q'$. By (Proc *Ch* Input), $E \vdash x(y_1, \ldots, y_k).Q'$.

In the (Proc *Un* Input) case, we have that $E \vdash x : Un$, $E, y_1{:}Un, \ldots, y_n{:}Un \vdash P'$. By induction hypothesis, $E, y_1{:}Un, \ldots, y_k{:}Un \vdash Q'$. By (Proc *Ch* Input), $E \vdash x(y_1, \ldots, y_k).Q'$.

Part (2) is symmetric.

(Struct Par Zero) Then $P = P' \mid \mathbf{0}$ and $Q = P'$.

For (1), assume $E \vdash P' \mid \mathbf{0}$. This must have been derived from (Proc Par) with $E \vdash P'$.

For (2), assume $E \vdash P'$. By Lemma 32, $E \vdash \diamond$. By (Proc Zero), $E \vdash \mathbf{0}$ By (Proc Par), $E \vdash P' \mid \mathbf{0}$

(Struct Par Comm) Then $P = P' \mid P''$ and $Q = P'' \mid P'$.

For (1), assume $E \vdash P' \mid P''$. This must have been derived from $E \vdash P'$ and $E \vdash P''$. By (Proc Par), $E \vdash P'' \mid P'$. Hence, $E \vdash Q$.

Part (2) is symmetric.

(Struct Par Assoc) Then $P = (P' \mid P'') \mid P'''$ and $Q = P' \mid (P'' \mid P''')$.

For (1), assume $E \vdash (P' \mid P'') \mid P'''$. This must have been derived from (Proc Par) twice, with $E \vdash P'$, $E \vdash P''$, and $E \vdash P'''$. By (Proc Par) twice, $E \vdash P' \mid (P'' \mid P''')$. Hence $E \vdash Q$.

Part (2) is symmetric.

(Struct Repl Par) Then $P = !P'$ and $Q = P' \mid !P'$. For (1), assume $E \vdash !P'$. This must have been derived from (Proc Repl), with $E \vdash P'$. By (Proc Par),

$E \vdash P' \mid !P'$. Hence, $E \vdash Q$.

For (2), assume $E \vdash P' \mid !P'$. This must have been derived from (Proc Par), with $E \vdash P'$ and $E \vdash !P'$. Hence, $E \vdash P$.

(Struct Res Res) In this case we have $P = (\nu x_1)(\nu x_2)P'$ and $Q = (\nu x_2)(\nu x_1)P'$ with $x_1 \neq x_2$.

For (1), assume $E \vdash (\nu x_1)(\nu x_2)P'$. This must have been derived from (Proc Res) twice, with $E, x_1{:}T_1, x_2{:}T_2 \vdash P'$, for some T_1, T_2. By Lemma 33, we have $E, x_2{:}T_2, x_1{:}T_1 \vdash P'$. By (Proc Res) twice we have $E \vdash (\nu x_2)(\nu x_1)P'$. Part (2) is symmetric.

(Struct Res Par) Then $P = (\nu x)(P' \mid P'')$ and $Q = P' \mid (\nu x)P''$, with $x \notin \mathit{fn}(P')$.

For (1), assume $E \vdash P$. This must have been derived from (Proc Res), with $E, x{:}T \vdash P' \mid P''$, for some T, and from (Proc Par), with $E, x{:}T \vdash P'$ and $E, x{:}T \vdash P''$. By Lemma 34, since $x \notin \mathit{fn}(P')$, we have $E \vdash P'$. By (Proc Res) we have $E \vdash (\nu x)P''$. By (Proc Par) we have $E \vdash P' \mid (\nu x)P''$, that is, $E \vdash Q$.

For (2), assume $E \vdash Q$. This must have been derived from (Proc Par), with $E \vdash P'$ and $E \vdash (\nu x)P''$, and from (Proc Res), with $E, x{:}T \vdash P''$, for some T. By Lemma 17, $E, x{:}T \vdash P'$. By (Proc Par), $E, x{:}T \vdash P' \mid P''$. By (Proc Res), $E \vdash (\nu x)(P' \mid P'')$, that is, $E \vdash P$. $\square$

Restatement of Proposition 8 *If $E \vdash P$ and $P \to Q$ then $E \vdash Q$.*

Proof By induction on the derivation of $P \to Q$.

(Red I/O) Then $P = \overline{x}\langle y_1, \ldots, y_k\rangle \mid x(z_1, \ldots, z_k).P'$ and $Q = P'\{z_1 \leftarrow y_1\} \cdots \{z_k \leftarrow y_k\}$. Assume $E \vdash P$. This must have been derived from (Proc Par) with $E \vdash x(z_1, \ldots, z_k).P'$ and $E \vdash \overline{x}\langle y_1, \ldots, y_k\rangle$. The former must have been derived either from (Proc *Ch* Input) or from (Proc *Un* Input). In the (Proc *Ch* Input) case, we have that $E \vdash x : \mathit{Ch}[T_1, \ldots, T_k]$, for some $T_1, \ldots, T_k$, and $E, z_1{:}T_1, \ldots, z_k{:}T_k \vdash P'$. $E \vdash x : \mathit{Ch}[T_1, \ldots, T_k]$, and Lemma 36, imply that the latter judgment $E \vdash \overline{x}\langle y_1, \ldots, y_k\rangle$ must have been derived from (Proc *Ch* Output), with $E \vdash y_i : T_i$ for each $i \in 1..k$. By k applications of Lemma 11, we get $E \vdash P'\{z_1 \leftarrow y_1\} \cdots \{z_k \leftarrow y_k\}$.

In the (Proc *Un* Input) case, we have $E \vdash x : \mathit{Un}$, and $E, z_1{:}\mathit{Un}, \ldots, z_k{:}\mathit{Un} \vdash P'$. $E \vdash x : \mathit{Un}$, and Lemma 36, imply that the latter judgment $E \vdash \overline{x}\langle y_1, \ldots, y_k\rangle$ must have been derived from (Proc *Un* Output), with $E \vdash y_i : \mathit{Un}$ for each $i \in 1..k$. By k applications of Lemma 11, we get $E \vdash P'\{z_1 \leftarrow y_1\} \cdots \{z_k \leftarrow y_k\}$.

(Red Par) Here $P = P' \mid R$ and $Q = Q' \mid R$ with $P' \to Q'$. Assume $E \vdash P$. This must have been derived using (Proc Par) from $E \vdash P'$ and $E \vdash R$. By induction hypothesis, $E \vdash Q'$. By (Proc Par), $E \vdash Q' \mid R$, that is, $E \vdash Q$.

(Red Res) Here $P = (\nu x)P'$ and $Q = (\nu x)Q'$ with $P' \to Q'$. Assume $E \vdash P$. This must have been derived using (Proc Res) from $E, x{:}T \vdash P'$, for some T. By induction hypothesis, $E, x{:}T \vdash Q'$. By (Proc Res), $E \vdash (\nu x)Q'$, that is, $E \vdash Q$.

(Red $\equiv$) Here $P \equiv P'$, $P' \to Q'$, and $Q' \equiv Q$. Assume $E \vdash P$. By Lemma 7, $E \vdash P'$. By induction hypothesis, $E \vdash Q'$. By Lemma 7, $E \vdash Q$. $\square$

D Facts Needed in Proof of the Secrecy Theorem

This appendix contains proofs of Proposition 9 and Proposition 10, used in the proof of Theorem 1, together with several auxiliary lemmas.

Lemma 37 *Suppose $E \vdash P$.*

(1) *If $P \xrightarrow{x} (y_1, \ldots, y_n)Q$ and $E \vdash x : Un$ and $E \vdash y'_i : Un$ for each $i \in 1..n$ then $E \vdash Q\{y_1 \leftarrow y'_1\} \cdots \{y_n \leftarrow y'_n\}$.*

(2) *If $P \xrightarrow{\overline{x}} (\nu z_1, \ldots, z_m)\langle y_1, \ldots, y_n\rangle Q$ and $E \vdash x : Un$ and $\{z_1, \ldots, z_m\} \cap dom(E) = \varnothing$ then $E, z_1{:}Un, \ldots, z_m{:}Un \vdash y_i : Un$ for each $i \in 1..n$, and $E, z_1{:}Un, \ldots, z_m{:}Un \vdash Q$.*

Proof Suppose $E \vdash P$.

(1) By definition, $P \xrightarrow{x} (y_1, \ldots, y_n)Q$ means that the names $y_1, \ldots, y_n$ are pairwise distinct, and there are processes P_1 and P_2 and pairwise distinct names $z_1, \ldots, z_m$ with $P \equiv (\nu z_1, \ldots, z_m)(x(y_1, \ldots, y_n).P_1 \mid P_2)$ and $Q \equiv (\nu z_1, \ldots, z_m)(P_1 \mid P_2)$ where $x \notin \{z_1, \ldots, z_m\}$ and $\{y_1, \ldots, y_n\} \cap (\{z_1, \ldots, z_m\} \cup fn(P_2)) = \varnothing$. Since the names $z_1, \ldots, z_m$ are bound, we may assume that $\{z_1, \ldots, z_m\} \cap \{y'_1, \ldots, y'_n\} = \varnothing$. By Lemma 31, this and $\{y_1, \ldots, y_n\} \cap (\{z_1, \ldots, z_m\} \cup fn(P_2)) = \varnothing$, imply that $Q\{y_1 \leftarrow y'_1\} \cdots \{y_n \leftarrow y'_n\} \equiv (\nu z_1, \ldots, z_m)(P_1\{y_1 \leftarrow y'_1\} \cdots \{y_n \leftarrow y'_n\} \mid P_2)$. By Lemma 7, $E \vdash P$ and $P \equiv (\nu z_1, \ldots, z_m)(x(y_1, \ldots, y_n).P_1 \mid P_2)$ imply that $E \vdash (\nu z_1, \ldots, z_m)(x(y_1, \ldots, y_n).P_1 \mid P_2)$. Hence, there are types $T_1, \ldots, T_m$ with $E' \vdash x(y_1, \ldots, y_n).P_1$ and $E' \vdash P_2$ where $E' = E, z_1{:}T_1, \ldots, z_m{:}T_m$. By assumption and Lemma 35, $E' \vdash x : Un$. So, only (Proc *Un* Input) can derive $E' \vdash x(y_1, \ldots, y_n).P_1$. Hence, we have $E'' \vdash P_1$ where $E'' = E', y_1{:}Un, \ldots, y_n{:}Un$. By assumption, $E \vdash y'_i : Un$ for each $i \in 1..n$. Hence, by Lemmas 35 and 11 we get that $E' \vdash P_1\{y_1 \leftarrow y'_1\} \cdots \{y_n \leftarrow y'_n\}$. By (Proc Par) and (Proc Res), this and $E' \vdash P_2$ imply that $E \vdash (\nu z_1, \ldots, z_m)(P_1\{y_1 \leftarrow y'_1\} \cdots \{y_n \leftarrow y'_n\} \mid P_2)$. Finally, by Lemma 7 we get $E \vdash Q\{y_1 \leftarrow y'_1\} \cdots \{y_n \leftarrow y'_n\}$.

(2) By definition, $P \xrightarrow{\overline{x}} (\nu z_1, \ldots, z_m)\langle y_1, \ldots, y_n\rangle Q$ means that the names $z_1, \ldots, z_m$ are pairwise distinct with $P \equiv (\nu z_1, \ldots, z_m)(\overline{x}\langle y_1, \ldots, y_n\rangle \mid Q)$ where $x \notin \{z_1, \ldots, z_m\}$ and $\{z_1, \ldots, z_m\} \subseteq \{y_1, \ldots, y_n\}$. By Lemma 7, $E \vdash P$ and $P \equiv (\nu z_1, \ldots, z_m)(\overline{x}\langle y_1, \ldots, y_n\rangle \mid Q)$ imply that $E \vdash (\nu z_1, \ldots, z_m)(\overline{x}\langle y_1, \ldots, y_n\rangle \mid Q)$. Given this and $\{z_1, \ldots, z_m\} \cap dom(E) = \varnothing$, there are types $T_1, \ldots, T_m$ such that $E' \vdash \overline{x}\langle y_1, \ldots, y_n\rangle$ and $E' \vdash Q$

where $E' = E, z_1{:}T_1, \dots, z_m{:}T_m$. By assumption and Lemma 35, $E' \vdash x : \mathit{Un}$. Therefore only (Proc *Un* Input) can derive $E' \vdash \overline{x}\langle y_1, \dots, y_n\rangle$ with $E' \vdash y_i : \mathit{Un}$ for each $i \in 1..n$. Since $\{z_1, \dots, z_m\} \subseteq \{y_1, \dots, y_n\}$ it follows that $T_i = \mathit{Un}$ for each $i \in 1..m$. So $E' = E, z_1{:}\mathit{Un}, \dots, z_m{:}\mathit{Un}$, and therefore we have $E, z_1{:}\mathit{Un}, \dots, z_m{:}\mathit{Un} \vdash y_i : \mathit{Un}$ for each $i \in 1..n$, and $E, z_1{:}\mathit{Un}, \dots, z_m{:}\mathit{Un} \vdash Q$ as required. $\Box$

Lemma 38 *Consider any process P_0 and any set S_0. Suppose there is E_0 such that $E_0 \vdash P_0$ and $\mathit{dom}(E_0) = \mathit{fn}(P_0) \cup S_0$ and $E_0 \vdash x : \mathit{Un}$ for each $x \in S_0$. If $(P_0, S_0) \mathbb{R} (P', S')$ there is E' such that $E_0, E' \vdash P'$ and $\mathit{dom}(E_0, E') = \mathit{fn}(P_0) \cup S'$ and $E_0, E' \vdash x : \mathit{Un}$ for each $x \in S'$.*

Proof We assume there is E_0 such that $E_0 \vdash P_0$ and $\mathit{dom}(E_0) = \mathit{fn}(P_0) \cup S_0$ and $E_0 \vdash x : \mathit{Un}$ for each $x \in S_0$. We proceed by induction on the derivation of $(P_0, S_0) \mathbb{R} (P', S')$. There are four cases to consider.

(1) We have $(P_0, S_0) \mathbb{R} (P_0, S_0)$. Take $E' = \varnothing$ and by assumption we have that $E_0, E' \vdash P_0$ and $\mathit{dom}(E_0, E') = \mathit{fn}(P_0) \cup S_0$ and $E_0, E' \vdash x : \mathit{Un}$ for each $x \in S_0$.

(2) We have $(P_0, S_0) \mathbb{R} (P', S')$ from $(P_0, S_0) \mathbb{R} (P, S')$ and $P \to P'$. By induction hypothesis, $(P_0, S_0) \mathbb{R} (P, S')$ implies there is E' such that $E_0, E' \vdash P$ and $\mathit{dom}(E_0, E') = \mathit{fn}(P_0) \cup S'$ and $E_0, E' \vdash x : \mathit{Un}$ for each $x \in S'$. By Proposition 8, $P \to P'$ implies $E, E' \vdash P'$.

(3) We have $(P_0, S_0) \mathbb{R} (Q\{y_1{\leftarrow}z_1, \dots, y_n{\leftarrow}z_n\}, S \cup \{z_1, \dots, z_n\})$ derived from $(P_0, S_0) \mathbb{R} (P, S)$, $P \xrightarrow{x} (y_1, \dots, y_n)Q$, $x \in S$, and $(\{z_1, \dots, z_n\} - S) \cap \mathit{fn}(P_0) = \varnothing$. By induction hypothesis, $(P_0, S_0) \mathbb{R} (P, S)$ implies there is E such that $E_0, E \vdash P$ and $\mathit{dom}(E_0, E) = \mathit{fn}(P_0) \cup S$ and $E_0, E \vdash x' : \mathit{Un}$ for each $x' \in S$. Let $\{z'_1, \dots, z'_m\} = \{z_1, \dots, z_n\} - S$ and let $E' = E, z'_1{:}\mathit{Un}, \dots, z'_m{:}\mathit{Un}$. We have $\{z'_1, \dots, z'_m\} \cap \mathit{fn}(P_0) = \varnothing$. From $\mathit{dom}(E_0, E) = \mathit{fn}(P_0) \cup S$ it follows that $\{z'_1, \dots, z'_m\} \cap \mathit{dom}(E_0, E) = \varnothing$, and therefore that $E_0, E' \vdash \diamond$. By Lemma 35, this and $E_0 \vdash P$ imply $E_0, E' \vdash P$. Since $x \in S$, we have $E_0, E \vdash x : \mathit{Un}$, and hence by Lemma 35, that $E_0, E' \vdash x : \mathit{Un}$. By Lemma 37(1), $E_0, E' \vdash P$ and $P \xrightarrow{x} (y_1, \dots, y_n)Q$ and $E_0, E' \vdash x : \mathit{Un}$ and $E_0, E' \vdash z_i : \mathit{Un}$ for each $i \in 1..n$ then $E_0, E' \vdash Q\{y_1{\leftarrow}z_1\} \cdots \{y_n{\leftarrow}z_n\}$. We have $\mathit{dom}(E_0, E') = \mathit{dom}(E_0, E) \cup \{z'_1, \dots, z'_m\} = \mathit{fn}(P_0) \cup S \cup \{z'_1, \dots, z'_m\} = \mathit{fn}(P_0) \cup S \cup \{z_1, \dots, z_n\}$. Finally, we have $E_0, E' \vdash x' : \mathit{Un}$ for each $x' \in S \cup \{z_1, \dots, z_n\}$.

(4) We have $(P_0, S_0) \mathbb{R} (Q, S \cup \{y_1, \dots, y_n\})$ from $(P_0, S_0) \mathbb{R} (P, S)$ and $P \xrightarrow{\overline{x}} (\nu z_1, \dots, z_m)\langle y_1, \dots, y_n\rangle Q$ and $x \in S$ and $\{z_1, \dots, z_m\} \cap (S \cup \mathit{fn}(P_0)) = \varnothing$. By induction hypothesis, $(P_0, S_0) \mathbb{R} (P, S)$ implies there is E such that $E_0, E \vdash P$ and $\mathit{dom}(E_0, E) = \mathit{fn}(P_0) \cup S$ and $E_0, E \vdash x' : \mathit{Un}$ for each $x' \in S$. Let $E' = E, z_1{:}\mathit{Un}, \dots, z_m{:}\mathit{Un}$. By Lemma 35, we get that $E_0, E' \vdash x' : \mathit{Un}$ for each $x' \in S$. By Lemma 37(2), $P \xrightarrow{\overline{x}} (\nu z_1, \dots, z_m)\langle y_1, \dots, y_n\rangle Q$ and $E_0, E \vdash x : \mathit{Un}$

and $\{z_1, \ldots, z_m\} \cap dom(E_0, E) = \varnothing$ imply $E_0, E' \vdash y_i : Un$ for each $i \in 1..n$, and $E_0, E' \vdash Q$. It follows that $\{y_1, \ldots, y_n\} \subseteq dom(E_0, E') = fn(P_0) \cup S \cup \{z_1, \ldots, z_m\}$. Now, $P \xrightarrow{\overline{x}} (\nu z_1, \ldots, z_m)\langle y_1, \ldots, y_n\rangle Q$ implies $\{z_1, \ldots, z_m\} \subseteq \{y_1, \ldots, y_n\}$. So $fn(P_0) \cup S \cup \{z_1, \ldots, z_m\} = fn(P_0) \cup S \cup \{y_1, \ldots, y_n\}$. Hence, we have $dom(E_0, E') = fn(P_0) \cup S \cup \{z_1, \ldots, z_m\} = fn(P_0) \cup S \cup \{y_1, \ldots, y_n\}$. $\Box$

Restatement of Proposition 9 *Suppose that $\varnothing, y_1{:}Un, \ldots, y_n{:}Un, x{:}T \vdash P$ where $T \neq Un$. Then the process P preserves the secrecy of the name x from $\{y_1, \ldots, y_n\}$.*

Proof We may assume $x \in fn(P)$, or else vacuously P preserves its secrecy. Let $E = \varnothing, y_1{:}Un, \ldots, y_n{:}Un, x{:}T$ and $S = \{y_1, \ldots, y_n\}$ so that $E \vdash P$ and $dom(E) = fn(P) \cup S$ (since $x \in fn(P)$) and $E \vdash y : Un$ for each $y \in S$. Consider any process P' and S' such that $(P, S) \mathbb{R} (P', S')$. By Lemma 38, there is E' such that $E, E' \vdash P'$ and $dom(E, E') = fn(P_0) \cup S'$ and $E, E' \vdash y : Un$ for each $y \in S'$. We know that $E, E' \vdash x : T$ and $T \neq Un$. Therefore, it cannot be that $x \in S'$, or else we would get that $E, E' \vdash x : Un$, which by Lemma 36 is incompatible with $E, E' \vdash x : T$. Hence, the process P preserves the secrecy of the name x from $\{y_1, \ldots, y_n\}$. $\Box$

Lemma 39 *If $E \vdash \diamond$ then $[\![E]\!]_G \vdash \diamond$.*

Proof A routine induction on the derivation of $E \vdash \diamond$. $\Box$

Lemma 40 *If $E \vdash x : T$ then $[\![E]\!]_G \vdash x : [\![T]\!]_G$.*

Proof Since $E \vdash x : T$ can only derive using (Exp x), E takes the form $E', x{:}T, E''$ and $E \vdash \diamond$. By Lemma 39, $[\![E]\!]_G \vdash \diamond$. By definition, we have $[\![E]\!]_G = [\![E']\!]_G, x{:}[\![T]\!]_G, [\![E'']\!]_G$. So, by (Exp x), we get that $[\![E]\!]_G \vdash x : [\![T]\!]_G$. $\Box$

Restatement of Proposition 10 *If $E \vdash P$ then $[\![E]\!]_G \vdash erase(P)$.*

Proof By induction on the derivation of $E \vdash P$.

(Proc GRes) Then $E \vdash (\nu H)P$ derives from $E, H \vdash P$. By induction hypothesis, $[\![E, H]\!]_G \vdash erase(P)$. Since $[\![E, H]\!]_G = [\![E]\!]$ and $erase((\nu H)P) = erase(P)$, we obtain $[\![E]\!] \vdash erase((\nu H)P)$.

(Proc Res) Then $E \vdash (\nu x{:}T)P$ derives from $E, x{:}T \vdash P$. By induction hypothesis, $[\![E]\!]_G, x{:}[\![T]\!]_G \vdash erase(P)$. By (Proc Res), $[\![E]\!]_G \vdash (\nu x) erase(P)$. Since $erase((\nu x{:}T)P) = (\nu x) erase(P)$ we get that $[\![E]\!]_G \vdash erase((\nu x{:}T)P)$.

(Proc Par) Then $E \vdash P \mid Q$ derives from $E \vdash P$ and $E \vdash Q$. By induction hypothesis, $[\![E]\!]_G \vdash erase(P)$ and $[\![E]\!]_G \vdash erase(Q)$. By (Proc Par), $[\![E]\!]_G \vdash erase(P) \mid erase(Q)$. Since $erase(P \mid Q) = erase(P) \mid erase(Q)$ we get that $[\![E]\!]_G \vdash erase(P \mid Q)$.

(Proc Repl) Then $E \vdash\, !P$ derives from $E \vdash P$. By induction hypothesis,

$[\![E]\!]_G \vdash \mathit{erase}(P)$. By (Proc Repl), $[\![E]\!]_G \vdash\ !\mathit{erase}(P)$. Since $\mathit{erase}(!P) = !\mathit{erase}(P)$ we get that $[\![E]\!]_G \vdash \mathit{erase}(!P)$.

(Proc Input) Then $E \vdash x(y_1{:}T_1, \ldots, y_k{:}T_n).P$ derives from the judgments $E \vdash x : H[T_1, \ldots, T_n]$ and $E, y_1{:}T_1, \ldots, y_n{:}T_n \vdash P$. By Lemma 40, we get $[\![E]\!]_G \vdash x : [\![H[T_1, \ldots, T_n]]\!]_G$. By induction hypothesis, we get $[\![E, y_1{:}T_1, \ldots, y_n{:}T_n]\!]_G \vdash \mathit{erase}(P)$, that is, $[\![E]\!]_G, y_1{:}[\![T_1]\!]_G, \ldots, y_n{:}[\![T_n]\!]_G \vdash \mathit{erase}(P)$.

If $G \in \mathit{fg}(H[T_1, \ldots, T_n])$ then $[\![H[T_1, \ldots, T_n]]\!]_G = \mathit{Ch}[[\![T_1]\!]_G, \ldots, [\![T_n]\!]_G]$. Hence, by (Proc *Ch* Input) we obtain that $[\![E]\!]_G \vdash x(y_1, \ldots, y_n).P$.

Otherwise, we have $G \notin \mathit{fg}(H[T_1, \ldots, T_n])$ and hence $G \notin \mathit{fg}(T_i)$ for each $i \in 1..n$. Therefore, $[\![H[T_1, \ldots, T_n]]\!]_G = \mathit{Un}$ and also $[\![T_i]\!] = \mathit{Un}$ for each $i \in 1..n$. Hence, by (Proc *Un* Input) we obtain that $[\![E]\!]_G \vdash x(y_1, \ldots, y_n).P$.

(Proc Output) Then $E \vdash \overline{x}\langle y_1, \ldots, y_n\rangle$ derives from $E \vdash y_i : T_i$ for each $i \in 1..n$ and from $E \vdash x : H[T_1, \ldots, T_n]$. By Lemma 40, we get that $[\![E]\!]_G \vdash y_i : [\![T_i]\!]_G$ for each $i \in 1..n$ and $[\![E]\!]_G \vdash x : [\![H[T_1, \ldots, T_n]]\!]_G$.

If $G \in \mathit{fg}(H[T_1, \ldots, T_n])$ then $[\![H[T_1, \ldots, T_n]]\!]_G = \mathit{Ch}[[\![T_1]\!]_G, \ldots, [\![T_n]\!]_G]$. Hence, by (Proc *Ch* Output) we obtain that $[\![E]\!]_G \vdash \overline{x}\langle y_1, \ldots, y_n\rangle$.

Otherwise, we have $G \notin \mathit{fg}(H[T_1, \ldots, T_n])$ and hence $G \notin \mathit{fg}(T_i)$ for each $i \in 1..n$. Therefore, $[\![H[T_1, \ldots, T_n]]\!]_G = \mathit{Un}$ and also $[\![T_i]\!] = \mathit{Un}$ for each $i \in 1..n$. Hence, by (Proc *Un* Output) we get $[\![E]\!]_G \vdash \overline{x}\langle y_1, \ldots, y_n\rangle$. $\square$

Acknowledgements

We thank Rocco de Nicola, Roberto Gorrieri, Tony Hoare, and the anonymous reviewers for useful suggestions, and also Martín Abadi and Bruno Blanchet who pointed out an error in the conference version of this paper. Giorgio Ghelli was supported by Microsoft Research, and by "Ministero dell'Università e della Ricerca Scientifica e Tecnologica", project DATA-X.

References

[1] M. Abadi. Secrecy by typing in security protocols. *Journal of the ACM*, 46(5):749–786, September 1999.

[2] M. Abadi. Security protocols and specifications. In *Foundations of Software Science and Computation Structures (FOSSACS'99)*, volume 1578 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 1999.

[3] M. Abadi and B. Blanchet. Secrecy types for asymmetric communication. *Theoretical Computer Science*, 298(3):387–415, 2003.

[4] M. Abadi and A.D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148:1–70, 1999.

[5] C. Bodei, P. Degano, F. Nielson, and H. Riis Nielson. Control flow analysis for the π-calculus. In *Concurrency Theory (Concur'98)*, volume 1466 of *Lecture Notes in Computer Science*, pages 84–98. Springer, 1998.

[6] L. Cardelli, G. Ghelli, and A.D. Gordon. Secrecy and group creation. In C. Palamidessi, editor, *CONCUR 2000—Concurrency Theory*, volume 1877 of *Lecture Notes in Computer Science*, pages 365–379. Springer, 2000.

[7] L. Cardelli, G. Ghelli, and A.D. Gordon. Types for the ambient calculus. *Information and Computation*, 177:160–194, 2002.

[8] S. Dal Zilio and A.D. Gordon. Region analysis and a π-calculus with groups. *Journal of Functional Programming*, 12(3):229–292, 2002.

[9] D. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–242, 1976.

[10] D. Dolev and A.C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IC–29(12):198–208, 1983.

[11] M. J. Gabbay and A.M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 13:341–363, 2002.

[12] S. J. Gay. A sort inference algorithm for the polyadic pi-calculus. In *20th ACM Symposium on Principles of Programming Languages (POPL'93)*, 1993.

[13] R. Milner. The polyadic π-calculus: a tutorial. In *Proceedings of the International Summer School on Logic and Algebra of Specification, Marktoberdorf*, 1991. Available as Technical Report ECS–LFCS–91–180, Department of Computer Science, University of Edinburgh, October 1991. Reprinted in *Logic and Algebra of Specification*, ed. F. L. Bauer, W. Brauer, and H. Schwichtenberg, Springer, 1993.

[14] R. Milner. *Communicating and Mobile Systems: the π-Calculus.* Cambridge University Press, 1999.

[15] M. Odersky. Polarized name passing. In *Foundations of Software Technology and Theoretical Computer Science*, volume 1026 of *Lecture Notes in Computer Science*. Springer, 1995.

[16] B. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–454, 1996.

[17] J. Riely and M. Hennessy. A typed language for distributed mobile processes. In *25th ACM Symposium on Principles of Programming Languages (POPL'98)*, pages 378–390, 1998.

[18] D. Sangiorgi and D. Walker. *The π-calculus: A Theory of Mobile Processes.* Cambridge University Press, 2001.

[19] P. Sewell. Global/local subtyping and capability inference for a distributed π-calculus. In *25th International Colloquium on Automata, Languages, and Programming (ICALP'98)*, volume 1443 of *Lecture Notes in Computer Science*, pages 695–706. Springer, 1998.

[20] M. Tofte and J.-P. Talpin. Region-based memory management. *Information and Computation*, 132(2):109–176, 1997.

[21] V. T. Vasconcelos and K. Honda. Principal typing-schemes in a polyadic π-calculus. In *CONCUR'93—Concurrency Theory*, volume 715 of *Lecture Notes in Computer Science*, pages 524–538. Springer, 1993.