Secrecy and Group Creation

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

[1] Microsoft Research
[2] Pisa University

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.

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.

1.1 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.

1.2 Preventing Leakage

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 type-check:

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

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 [7]. 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 demand, 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 beacause, 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 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 type-checking 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 has 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 (for example, 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.

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

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 type-checked 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 type-check. The only important requirement is that the player must be type-checked 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 adapt a notion of secrecy due to Abadi to 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.

2 A Typed π-Calculus with Groups

We present here a typed π-calculus with groups and group creation. The earliest type system for the π-calculus, reported in Milner's book [10] but first published in 1991, is based on sorts; sorts are like groups in that each name belongs to a sort, but Milner's system has no construct for sort creation. Moreover, his system allows recursive definitions of sorts; we would need to add recursive types to our system to mimic such definitions. Subsequent type systems introduced a variety of channel type constructors and subtyping [11, 12].

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:

$T ::= G[T_1, \ldots, T_n]$	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; we write $P = Q$ to mean that P and Q are the same up to capture-avoiding renaming of bound variables.

Expressions and Processes:

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

$P \mid Q$	composition
$!P$	replication
$\mathbf{0}$	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, $P \to Q$, 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 communication channel.

Our rules for reduction and structural congruence are standard [10] apart from the inclusion of new rules for group creation, and the exclusion of garbage collection rules such as $\mathbf{0} \equiv (\nu x{:}T)\mathbf{0}$ and $x \notin fn(P) \Rightarrow (\nu x{:}T)P \equiv P$. Such rules are unnecessary for calculating reduction steps. In their presence, reduction can enlarge the set of free groups of a process. Hence, their inclusion 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)
$x \notin fn(P) \Rightarrow (\nu x{:}T)(P \mid Q) \equiv P \mid (\nu x{:}T)Q$	(Struct Res Par)
$(\nu G_1)(\nu G_2)P \equiv (\nu G_2)(\nu G_1)P$	(Struct GRes GRes)
$G \notin fg(T) \Rightarrow (\nu G)(\nu x{:}T)P \equiv (\nu x{:}T)(\nu G)P$	(Struct GRes Res)
$G \notin fg(P) \Rightarrow (\nu G)(P \mid Q) \equiv P \mid (\nu G)Q$	(Struct GRes Par)

Reduction: $P \to Q$

$\overline{x}\langle y_1,\ldots,y_n\rangle \mid x(z_1{:}T_1,\ldots,z_n{:}T_n).P \to P\{z_1{\leftarrow}y_1\}\cdots\{z_n{\leftarrow}y_n\}$	(Red I/O)
$P \to Q \Rightarrow P \mid R \to Q \mid R$	(Red Par)

$P \to Q \Rightarrow (\nu G)P \to (\nu G)Q$ (Red GRes)
$P \to Q \Rightarrow (\nu x{:}T)P \to (\nu x{:}T)Q$ (Red Res)
$P' \equiv P, P \to Q, Q \equiv Q' \Rightarrow P' \to Q'$ (Red $\equiv$)

The new rules for group creation are the congruence rules (Struct GRes) and (Red GRes), and the scope mobility rules (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; second, $E \vdash T$ means that T is well-formed 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.

Typing Rules:

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

(Type Chan) $\dfrac{G \in dom(E) \quad E \vdash T_1 \quad \cdots \quad E \vdash T_n}{E \vdash G[T_1, \ldots, T_n]}$ (Exp x) $\dfrac{E', x{:}T, E'' \vdash \diamond}{E', x{:}T, E'' \vdash x : T}$

(Proc GRes) $\dfrac{E, G \vdash P}{E \vdash (\nu G)P}$ (Proc Res) $\dfrac{E, x{:}T \vdash P}{E \vdash (\nu x{:}T)P}$ (Proc Zero) $\dfrac{E \vdash \diamond}{E \vdash \mathbf{0}}$ (Proc Par) $\dfrac{E \vdash P \quad E \vdash Q}{E \vdash P \mid Q}$ (Proc Repl) $\dfrac{E \vdash P}{E \vdash !P}$

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

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

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 a process of the form:

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

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 $T = G[\,]$. If that extrusion were allowed, that is, if we failed to prevent name clashes on group names, then the player could communicate with the opponent in a well-typed way, and secrecy would fail.

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

Proposition 1 (Subject Reduction). *If $E \vdash P$ and $P \rightarrow Q$ then $E \vdash Q$.*

Subject reduction allows us to prove secrecy properties like the following one.

Proposition 2. *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 are no processes P'' and P''' and a context $C[\,]$ such that $P \equiv (\nu G)(\nu x{:}G[T_1, \ldots, T_n])P''$, $P'' \rightarrow P'''$, $P''' \equiv C[\overline{p}\langle x\rangle]$, where p and x are not bound by $C[]$.*

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

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
$\quad (\nu x)P$	$\quad$ restriction
$\quad P \mid Q$	$\quad$ composition
$\quad !P$	$\quad$ replication
$\quad \mathbf{0}$	$\quad$ 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 reduction, $P \to Q$, and structural congruence, $P \equiv Q$, relations on untyped processes corresponding to the typed reduction and structural congruence relations. We omit the standard rules, which are obtained from the rules of the typed calculus by erasing type annotations and deleting rules that mention the new-group construct.

Erasures of type annotations and group restrictions:

$\mathit{erase}((\nu G)P) \triangleq \mathit{erase}(P)$	$\mathit{erase}((\nu x{:}T)P) \triangleq (\nu x)\mathit{erase}(P)$
$\mathit{erase}(\mathbf{0}) \triangleq \mathbf{0}$	$\mathit{erase}(P \mid Q) \triangleq \mathit{erase}(P) \mid \mathit{erase}(Q)$
$\mathit{erase}(!P) \triangleq\ !\mathit{erase}(P)$	$\mathit{erase}(\overline{x}\langle y_1, \ldots, y_n\rangle) \triangleq \overline{x}\langle y_1, \ldots, y_n\rangle$
$\mathit{erase}(x(y_1{:}T_1, \ldots, y_n{:}T_n).P) \triangleq x(y_1, \ldots, y_n).\mathit{erase}(P)$	

Proposition 3 (Erasure). *For all typed processes P and Q, and untyped processes R, $P \to Q$ implies $\mathit{erase}(P) \to \mathit{erase}(Q)$ and $\mathit{erase}(P) \to R$ implies there is a typed process Q such that $P \to Q$ 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 $P \xrightarrow{x} (y_1, \ldots, y_n)Q$ means that P is ready to receive an input tuple on the channel x in the variables $y_1, \ldots, y_n$, and then continue as Q. The variables $y_1, \ldots, y_n$ are bound with scope Q. An output transition $P \xrightarrow{\overline{x}} (\nu z_1, \ldots, z_m)\langle y_1, \ldots, y_n\rangle Q$ means that P is ready to transmit an output tuple $\langle y_1, \ldots, y_n\rangle$ on the channel x, and then continue as Q. The set $\{z_1, \ldots, z_m\} \subseteq \{y_1, \ldots, y_n\}$ consists of freshly generated names whose scope includes both the tuple $\langle y_1, \ldots, y_n\rangle$ and the process Q. The names $z_1, \ldots, z_n$ 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 $P \xrightarrow{x} (y_1, \ldots, y_n)Q$ if and only if the names $y_1, \ldots, y_n$ are pairwise distinct, and there are processes P_1 and P_2 and pairwise distinct names $z_1, \ldots, z_m$ such that $P \equiv (\nu z_1, \ldots, z_m)(x(y_1, \ldots, y_n).P_1 \mid P_2)$ and $Q \equiv (\nu z_1, \ldots, z_m)(P_1 \mid P_2)$ where $x \notin \{z_1, \ldots, z_m\}$, and $\{y_1, \ldots, y_n\} \cap (\{z_1, \ldots, z_m\} \cup \mathit{fn}(P_2)) = \varnothing$.
- Let $P \xrightarrow{\overline{x}} (\nu z_1, \ldots, z_m)\langle y_1, \ldots, y_n\rangle Q$ if and only if the names $z_1, \ldots, z_m$ are pairwise distinct, and we have $P \equiv (\nu z_1, \ldots, z_m)(\overline{x}\langle y_1, \ldots, y_n\rangle \mid Q)$ where $x \notin \{z_1, \ldots, z_m\}$ and $\{z_1, \ldots, z_m\} \subseteq \{y_1, \ldots, y_n\}$.

3.2 A Secrecy Theorem

The following definition is inspired by Abadi's definition of secrecy [2] for the untyped spi calculus [3]. Abadi attributes the underlying idea to Dolev and Yao [8]: 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, 3].

We model the external opponent simply by the finite set of names S known to it. We inductively define a relation $(P_0, S_0)\ \mathcal{R}\ (P, S)$ to mean that starting from a process P_0 and an opponent knowing S_0, we may reach a state in which P_0 has evolved into P, and the opponent now knows S.

(1) $(P_0, S_0)\ \mathcal{R}\ (P_0, S_0)$
(2) If $(P_0, S_0)\ \mathcal{R}\ (P, S)$ and $P \to Q$ then $(P_0, S_0)\ \mathcal{R}\ (Q, S)$.
(3) If $(P_0, S_0)\ \mathcal{R}\ (P, S)$, $P \xrightarrow{x} (y_1, \ldots, y_n)Q$, $x \in S$, and $(\{z_1, \ldots, z_n\} - S) \cap \mathit{fn}(P_0) = \varnothing$ then $(P_0, S_0)\ \mathcal{R}\ (Q\{y_1{\leftarrow}z_1, \ldots, y_n{\leftarrow}z_n\}, S \cup \{z_1, \ldots, z_n\})$.
(4) If $(P_0, S_0)\ \mathcal{R}\ (P, S)$, $P \xrightarrow{\overline{x}} (\nu z_1, \ldots, z_m)\langle y_1, \ldots, y_n\rangle Q$ and $x \in S$ and $\{z_1, \ldots, z_m\} \cap (S \cup \mathit{fn}(P_0)) = \varnothing$ then $(P_0, S_0)\ \mathcal{R}\ (Q, S \cup \{y_1, \ldots, y_n\})$.

Clause (1) says that (P_0, S_0) 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 $(\{z_1, \ldots, z_n\} - S) \cap \mathit{fn}(P_0) = \varnothing$ ensures these fresh names are not confused with names initially known by P_0. Clause (4) allows the process to output the tuple $\langle y_1, \ldots, y_n\rangle$ to the opponent, who then knows the names $S \cup \{y_1, \ldots, y_n\}$, provided the channel x is known to the opponent. The names $\{z_1, \ldots, z_n\}$ (included in $\{y_1, \ldots, y_n\}$) are freshly created by the process; the condition $\{z_1, \ldots, z_m\} \cap (S \cup \mathit{fn}(P_0)) = \varnothing$ ensures these fresh names are not confused with names currently known by the opponent or initially known by P_0.

Next, we give definitions of when a name is revealed to an opponent, and formalize the secrecy property of group creation discussed in Section 1.

Revealing Names, Preserving their Secrecy:

Suppose S is a set of names and P is a process.
Then P *may reveal* x *to* S if and only if there are P' and S' such that $(P, S)\ \mathcal{R}\ (P', S')$ and $x \in S'$; otherwise, P *preserves the secrecy of* x *from* S.
Moreover, $(\nu x)P$ *may reveal the restricted name* x *to* S if and only if there is a name $y \notin S \cup \mathit{fn}(P)$ such that $P\{x{\leftarrow}y\}$ may reveal y to S; otherwise $(\nu x)P$ *preserves the secrecy of the restricted name* x *from* S.

Theorem 1 (Secrecy). *Suppose that* $E \vdash (\nu G)(\nu x{:}T)P$ *where* $G \in \mathit{fg}(T)$. *Let* S *be the names occurring in* $\mathit{dom}(E)$. *Then the erasure* $(\nu x)\mathit{erase}(P)$ *of* $(\nu G)(\nu x{:}T)P$ *preserves the secrecy of the restricted name* x *from* S.

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

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.

Types:

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

For any 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 $Ch[[\![T_1]\!]_G, \ldots, [\![T_n]\!]_G]$. 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.

Next, we define the three judgments of the auxiliary type system: first, $E \vdash \diamond$ means that E is well-formed; second, $E \vdash x : T$ means that $x{:}T$ is in E, and that E is well-formed; and, third, $E \vdash P$ means that P is well-formed in the environment E.

Typing Rules:

$$\frac{}{\varnothing \vdash \diamond} \qquad \frac{E \vdash \diamond \quad x \notin dom(E)}{E, x{:}T \vdash \diamond} \qquad \frac{E', x{:}T, E'' \vdash \diamond}{E', x{:}T, E'' \vdash x : T}$$

$$\frac{E, x{:}T \vdash P}{E \vdash (\nu x)P} \qquad \frac{E \vdash \diamond}{E \vdash \mathbf{0}} \qquad \frac{E \vdash P \quad E \vdash Q}{E \vdash P \mid Q} \qquad \frac{E \vdash P}{E \vdash\, !P}$$

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

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

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

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

The auxiliary type system is defined on untyped processes. Any untrusted opponent may be type-checked, as follows. This property makes the type system suitable for reasoning about processes containing both trusted and untrusted subprocesses.

Lemma 2. *For all P, if $fn(P) = \{x_1, \ldots, x_n\}$ then $\varnothing, x_1{:}Un, \ldots, x_n{:}Un \vdash P$.*

Structural congruence and reduction preserve typings.

Lemma 3. *If $E \vdash P$ and either $P \equiv Q$ or $P \to Q$ then $E \vdash Q$.*

The following fact is the crux of the proof of Theorem 1: an opponent who knows only untrusted names cannot learn any trusted one.

Proposition 4. *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 $S = \{y_1, \ldots, y_n\}$.*

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:

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

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

Lemma 4. *If $E \vdash P$ then $[\![E]\!]_G \vdash erase(P)$.*

Finally, we outline the proof of Theorem 1.

Lemma 5. *If $E, x{:}T, E' \vdash P$ and $E \vdash y : T$ then $E, E' \vdash P\{x{\leftarrow}y\}$.*

Lemma 6. *Let S be a finite set of names and P a process. Then P preserves the secrecy of x from S if and only if for all P', S', $(P, S)\ \mathcal{R}\ (P', S')$ implies that $x \notin S'$. Moreover, $(\nu x)P$ preserves the secrecy of the restricted name x from S if and only if for all $y \notin fn(P) \cup S$, $P\{x{\leftarrow}y\}$ preserves the secrecy of y from S.*

Proof of Theorem 1 *Suppose that $E \vdash (\nu G)(\nu x{:}T)P$ where $G \in fg(T)$. Let S be the names occurring in $dom(E)$. Then the erasure $(\nu x)erase(P)$ of $(\nu G)(\nu x{:}T)P$ preserves the secrecy of the restricted name x from S.*

Proof. Since the name x is bound, we may assume that $x \notin S$. Consider any name $y \notin fn(P) \cup S$. By Lemma 6, it is sufficient to show that $erase(P)\{x{\leftarrow}y\}$ preserves the secrecy of y from S. 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 $S = \{z_1, \ldots, z_n\}$. Lemma 4 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 4 implies that $erase(P)$ preserves the secrecy of x from S. If $x = y$ we are done, since in that case $erase(P)\{x{\leftarrow}y\} = erase(P)$. Otherwise, suppose $x \neq y$. Since $y \notin \{z_1, \ldots, z_n\}$ we can derive $\varnothing, z_1{:}Un, \ldots, z_n{:}Un$,

$y{:}[\![T]\!]_G, x{:}[\![T]\!]_G \vdash \mathit{erase}(P)$ using a weakening lemma, and also derive $\varnothing, z_1{:}Un, \ldots, z_n{:}Un, y{:}[\![T]\!]_G \vdash y : [\![T]\!]_G$. By the substitution lemma, Lemma 5, these two judgments imply $\varnothing, z_1{:}Un, \ldots, z_n{:}Un, y{:}[\![T]\!]_G \vdash \mathit{erase}(P)\{x{\leftarrow}y\}$. Hence, Proposition 4 implies that $\mathit{erase}(P)\{x{\leftarrow}y\}$ preserves the secrecy of the name y from $S = \{z_1, \ldots, z_n\}$. $\square$

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 [5]. 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. Another paper [6] extends the typed π-calculus of Section 2 with an effect system. That paper establishes a formal connection between group creation and the *letregion* construct of Tofte and Talpin's region-based memory management [15]. That paper generalizes our subject congruence, subject reduction, and erasure results (Lemma 1, Propositions 1 and 3) to the system of types and effects for the π-calculus. We conjecture that the main secrecy result of this paper, Theorem 1, would hold also for the 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 2. 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 [4] 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 group creation operator of this paper. Another is that an algorithm is known for computing flow analyses for the π-calculus, but we have not investigated algorithmic aspects of our type system. It would be interesting to consider whether algorithms for Milner's sort systems [9, 16] extend to our calculus.

Other related work on the π-calculus includes type systems for guaranteeing locality properties [13, 14]. These systems can enforce by type-checking that a name cannot be leaked 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.

Acknowledgements We thank Rocco de Nicola, Roberto Gorrieri, Tony Hoare, and the anonymous referees for useful suggestions. 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 *Proceedings FOSSACS'99*, volume 1578 of *LNCS*, pages 1–13. Springer, 1999.
3. M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148:1–70, 1999.
4. C. Bodei, P. Degano, F. Nielson, and H. Nielson. Control flow analysis for the π-calculus. In *Proceedings Concur'98*, volume 1466 of *LNCS*, pages 84–98. Springer, 1998.
5. L. Cardelli, G. Ghelli, and A. D. Gordon. Ambient groups and mobility types. In *Proceedings TCS2000*, LNCS. Springer, 2000. To appear.
6. S. Dal Zilio and A. D. Gordon. Region analysis and a π-calculus with groups. In *Proceedings MFCS2000*, LNCS. Springer, 2000. To appear.
7. D. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–242, 1976.
8. D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IC–29(12):198–208, 1983.
9. S. J. Gay. A sort inference algorithm for the polyadic pi-calculus. In *Proceedings POPL'93*. ACM, 1993.
10. R. Milner. *Communicating and Mobile Systems: the π-Calculus*. CUP, 1999.
11. M. Odersky. Polarized name passing. In *Proc. FST & TCS*, LNCS. Springer, December 1995.
12. B. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–454, 1996.
13. J. Riely and M. Hennessy. A typed language for distributed mobile processes. In *Proceedings POPL'98*, pages 378–390. ACM, 1998.
14. P. Sewell. Global/local subtyping and capability inference for a distributed π-calculus. In *Proceedings ICALP'98*, volume 1443 of *LNCS*, pages 695–706. Springer, 1998.
15. M. Tofte and J.-P. Talpin. Region-based memory management. *Information and Computation*, 132(2):109–176, 1997.
16. V. T. Vasconcelos and K. Honda. Principal typing-schemes in a polyadic π-calculus. In *Proceedings Concur'93*, volume 715 of *LNCS*, pages 524–538. Springer, 1993.