Proving Trust in Systems of Second-Order Processes (Extended Abstract)

Mads Dam
Swedish Institute of Computer Science
S-164 28 Kista, Sweden

Abstract

We consider the problem of proving correctness properties for concurrent systems with features such as higher-order communication and dynamic resource generation. As examples we consider operational models of security and authentication protocols based on the higher-order π-calculus. Key features such as nonces/time stamps, encryption/decryption, and key generation are modelled using channel name generation and second-order process communication. A temporal logic based on the modal μ-calculus is used to express secrecy and authenticity. Extensions include function space constructions to deal with process input and output. Contravariant recursion can be dealt with in two different ways, of which one, an iterative solution, is discussed in the paper. We propose a predicate of trust in a monotonically increasing set of channels as an example, and establish structural decomposition principles for this predicate for concurrent composition and local channel declaration. On this basis a type system for trust inference can be derived quite easily.

1 Introduction

In this paper we consider the problem of proving correctness properties in semantically very rich models of concurrent systems with features for communication of second- and higher-order objects (eg.: code), and for dynamically generating and communicating resource names. A long list of recent programming languages and models, including Java, CML, Facile, Oz, Actors, Erlang, and the π-calculus, explore these sorts of features to various extents. Typical of many applications written in these kinds of languages is that they are *open*, designed and intended to operate in environments that are possibly hostile, and at any rate only partially known at compile time. An important task is therefore to protect information and resources against intrusion, intended or otherwise. Intruders have at their disposal the full armoury usually considered in the field of computer security: they can steal messages, tamper with messages, crack codes, synthesize messages, store and replay messages, and much more. In the presence of higher-order communication they can even generate programs (viruses) that will be activated dynamically by the receiving agent. The question we address is how, in spite of this, we can prove that a system nonetheless performs correctly.

Key Management Protocols Classical key management protocols such as the Needham-Schroeder protocol are examples of programs designed to work reliably in face of hostile intruders. In this paper we show how key management protocols can fruitfully be viewed as higher-order communicating processes and we show some initial ideas as to how, on this basis, they can be verified. The idea is best introduced through a simple example, a modified version of the corrected version of the Andrew remote RPC protocol as introduced by Needham, Abadi, and Burrows [6]. This protocol is extremely simple, yet it introduces all the features needed to account for a whole class of key management protocols. Initially two participants, A and B (sometimes called `Alice` and `Bob`), share a private key K_{ab}. The task is to agree on a new session key. Using standard notation the protocol can be described as the exchange between A and B of the following three message sequence:

1. $A \to B : \{N_a\}_{K_{ab}}$
2. $B \to A : \{N_a, K'_{ab}\}_{K_{ab}}$
3. $A \to B : \{N_a\}_{K'_{ab}}$

In step (1) A transmits the nonce N_a to B, signalling his wish for B to generate a new session key. B responds, in step (2), by returning the new session key K'_{ab}, along with N_a to authenticate the message, both encrypted using the old key K_{ab}. In step (3), A returns the nonce N_a to B, this time encrypted using the new key K'_{ab}, serving as an acknowledgement to B that the previous message was received and decrypted.

Modelling Key Management Protocols as Second-Order Processes The protocol involves the following features:

1. Message passing and data type operations: Given a (possibly composite) message m it is possible to communicate m from A to B. It is also possible, given messages m_1 and m_2, to form the pair (m_1, m_2).

2. Private key encryption and decryption: Given a message m and a private key K, we can form $\{m\}_K$, m encrypted using key K. Also, given $\{m\}_K$ and given K we can decrypt to extract the message m.

3. Key generation: B has the capability of generating a new private key K'_{ab}, by assumption distinct from any other key known to any other participant in the exchange, friendly or hostile.

4. Nonce generation. It is possible to generate a fresh, non-composite piece of information, by assumption distinct from any other such pieces of information possessed by other participants.

We propose accounting for these features in the following fashion:

- Nonces and keys are names as in the π-calculus [11]. New names are declared by the binding $\nu a.A$ introducing a as a new name with scope initially extending over A but not further. In $\nu a.A$, a will by definition be distinct from any other name occurring freely in A. Furthermore it is possible dynamically to extend the scope of a by "scope extrusion", eg. $\nu a.b.[a]P$ which declares a new name a and immediately passes it to the outside world along the channel b.

- Encryption is second-order process passing. Names, hence keys, are channel identifiers. Hence a message m encrypted using the private key K can be regarded as an object that can deliver m to anyone happening to know K. That is, it is a process with one output port K along which m is passed to whoever possesses K and is willing to listen. Thus $\{m\}_K$ is in our notation identified with the process $K.[m]0$, the process which transmits m along K and then terminates.

This suggests using a second-order version of the π-calculus as a semantical framework for modelling key management protocols, and indeed this is what we propose to do. Of course such models will be highly idealised: For instance the bit lengths used to represent nonces and keys are bounded, opening up for attacks on the encryption/decryption algorithms, and information can often be extracted from encrypted messages with very limited knowledge of the keys. Nonetheless we believe that an idealised modelling of key management aspects alone can be useful, leaving analysis of actual encryption algorithms to be addressed by other means, even while recognising that a really water-tight boundary between the two is not a reasonable hope.

Nested Encryption and Firewalling One complication needs to be attended to, though. As encryption can be nested we need to consider process terms of the shape

$$P = a.[K_1.[K_2.[m]0].0]P',$$

modelling a process passing $\{\{m\}_{K_2}\}_{K_1}$ along a and then proceeding to act as P'. A process receiving such a packet and decrypting to extract m would have the shape

$$Q = a.\lambda X_1.(X_1 \mid (K_1.\lambda X_2.X_2 \mid (K_2.\lambda m.Q'(m)))).$$

That is, it receives the process X_1, activates it and tries to receive from it along K_1 another process, X_2, which it proceeds to activate, to try and receive from it m along K_2. As it stands, however, the π-calculus has no good way of preventing a third party from stealing m using K_2 once Q has decrypted using K_1. That is, once Q has reached the configuration $(K_2.[m]0) \mid (K_2.\lambda m.Q'(m))$, if an external intruder is present that may know about K_2 it will have the capability of receiving the m without necessarily having to know K_1 first. Thus decryption is unsafe, contrary to most reasonable modelling assumptions (cf. [6]).

A good fix is to use a firewalling, or blocking operator $A\backslash a$ preventing communication along the channel a between A and its environment. This operator is well known: It is just the CCS restriction operator, extended to the π-calculus in the obvious fashion by allowing state transitions $P\backslash a \stackrel{b}{\rightarrow} A$ just in case A has the shape $A'\backslash a$, $P \stackrel{b}{\rightarrow} A'$, and b, the communication channel, is distinct from a. This operator was also considered in the context of higher order processes by Thomsen [13]. Using the blocking operator we can protect m from theft along K_2 by putting $(K_2.[m]0) \mid (K_2.\lambda m.Q'(m))$ inside a K_2 firewall. Observe that we regard a as free in $A\backslash a$. Thus communication of a across the a firewall itself will be perfectly legitimate.

Specification in Second-Order Temporal Logic Our aim is to use a second-order temporal logic to specify desired correctness properties like secrecy and authenticity. Specifically we suggest using a fragment of the modal μ-calculus extended with first-order features for names and name generation, and two arrows to account for process input and output. The logic follows quite closely ideas put forward in [3], using a function arrow $\phi \to \psi$ for input dependency, and a second-order arrow $(\phi \to \psi) \to \gamma$ for contextual output dependency. The idea is the following: A process waiting to input a parameter x to continue acting as P is written as a lambda-abstraction $\lambda x.P$. A process wanting to output to some receiver the process Q_1 to continue acting as Q_2 is written as the term $[Q_1]Q_2$, called a *process concretion*. Sometimes Q_1 and Q_2 may share a vector of private channel names $\vec{a}$ wishing to maintain these connections after Q_1 has been passed to its receiver. Such a process concretion is written $\nu\vec{a}.[Q_1]Q_2$. Matching receiver and sender results in the term $\lambda x.P \mid \nu\vec{a}.[Q_1]Q_2$ which is identified with the term $\nu\vec{a}.\{Q_1/x\}P \mid Q_2$, alpha-converting the bound names $\vec{a}$ as needed to avoid collision with names free in P. The input arrow expresses the expected functional dependency: For $\lambda x.P$ to have the property $\phi \to \psi$ it must be the case that Q_1 has the property ϕ only if $\{Q_1/x\}P$ has the property ψ. The output arrow expresses dependency upon receiving context: The process concretion $\nu\vec{a}.[Q_1]Q_2$ will have the property $(\phi \to \psi) \to \gamma$ just in case $\lambda x.P$ has the property $\phi \to \psi$ only if $\nu\vec{a}.\{Q_1/x\}P \mid Q_2$ has the property γ. In [3] we showed how this setting could be used to achieve an appropriate level of discriminatory power when measured against a strong version of bisimulation equivalence, and we began investigating proof principles for these connectives.

Handling Contravariant Recursion Unfortunately contravariant recursion appears indispensable for formalising the trustedness predicates we have in mind. We have so far found no way around this difficulty. The problem is that the Knaster-Tarski fixed point theorem usually appealed to for least and greatest fixed point semantics require monotonicity which fails in the presence of contravariant recursion. In this extended abstract we abandon the standard fixed point semantics for an iterative construction. This semantics provides an induction principle which is used heavily in subsequent proofs. In the full version of the paper we give also an equivalent fixed point semantics based on intervals. The iterative construction exploits a continuity property which holds only for the fragment of the modal μ-calculus lacking least fixed points and diamonds (existential next-state quantifiers). This, however, is a limited loss in view of the nature of the properties in which we have primary interest: Matters like secrecy and authenticity would be expected to have formulations as invariants and not to use existential computation path quantification.

Proving Trust We then suggest a process predicate expressing trust in a monotonically increasing set of channels, using contravariant recursion and greatest fixed points only. The rest of the paper is devoted to proof principles for this trustedness predicate, and the proof of trust for a very simple example protocol. The most involved proof principles concern, as one should expect, parallel composition and name scoping. Several crucial lemmas need to be proved, of which we highlight two. First we need to show that if P is a process which respects trust of $\vec{a}$, and b does not occur freely in P, then P will respect the trust of $\vec{a} \cup \{b\}$ (we usually write $\vec{a}, b$ as a shorthand). The proof of this is, as one should expect, a simple inductive argument, using the induction principle hinted to earlier. However, we also need a corresponding result for functions, that if $\lambda x.P$ has the property $\vec{a}$ `trusted` $\to \vec{a}$ `trusted`, and if b does not occur freely in $\lambda x.P$, then $\lambda x.P$ will also have the property $\vec{a}, b$ `trusted` $\to \vec{a}, b$ `trusted`. This property has a far more intensional character as it has to do with definability of functions, and the proof is also much more delicate.

Deriving a Type System Having proved the crucial lemmas for decomposing trust for parallel compositions and ν declarations we show how we quite simply can derive a type system for proving assertions of the shape $\Gamma \vdash^{\vec{a}} A : \vec{b}$ `trusted` expressing that, under the assumptions Γ, and in the local context $\vec{a}$, A has the property $\vec{b}$ `trusted`.

Related Work The present paper can be viewed as part of an ongoing trend towards operationally based accounts of security and authentication protocol. The closest predecessor of this work is Abadi and Gordon's work on the spi calculus [2]. It is the credit of Abadi and Gordon first to have observed the usefulness of the π-calculus name scoping discipline for modelling security protocol features like nonce and key generation. In the spi calculus extra operators for encryption and decryption are added to the π-calculus. Properties such as secrecy and authenticity are accounted for in equational terms, for instance by reflecting insensitivity of environments to changes in trusted values. By con-

trast we represent such properties directly, as a logical formula. Moreover, due to the explicit treatment of encryption and decryption a rather non-standard version of testing equivalence has to be appealed to for the correctness proofs in [2]. This complication does not arise in our approach since we reduce encryption and decryption to more general computational features.

Recently a number of authors have attempted to use state exploration methods to analyse security protocols (cf. [10, 9]). In approaches like these the main difficulty is to faithfully represent protocols and intruders as finite state automata. Instead of leaving intruders undetermined, as in our approach, it becomes necessary to state explicitly at every possible step whether an action is or is not possible, including history dependencies. Secondly it becomes difficult to deal with unbounded information, such as protocols runs that can cause an in principle unbounded number of nonces, time stamps, or keys to be generated. For this reason (and for sheer model size considerations, one suspects) work has so far focused on public key encryption, and on single session establishment runs.

In another related strand of work a large number of authors have used static analysis and type systems to analyse security of information flow, cf. [1, 5, 8, 14]. The scope of these analyses is roughly the same as ours: They analyse whether security levels are respected during program execution, sometimes stratifying the analysis by eg. distinguishing readers and writers. As in our work revocation of trust is not supported. Our contribution to this line of work is to show how a type system for secure information flow can quite easily be derived from the very general and sound semantical basis that we provide, using the account of programs as second order π-calculus processes, and types/properties as interpreted second-order temporal formulas.

For the full version of the paper we refer the reader to `ftp://ftp.sics.se/pub/fdt/mfd/ptssop.ps.Z`.

2 Processes

In this section we give an informal presentation of the language used to model protocols, and as much of its operational semantics as is needed to understand the specification logic and the reasoning of the correctness proofs. Roughly, the process language is a merge of the π-calculus [11] with the second-order process-passing calculus presented in [3]. It uses the following primitive objects:

- Channel names a, b, along with the special label τ, used for invisible, or silent transitions.
- Agent variables x, y.
- Agent constants D. With each constant is associated a unique defining equation $D = A$ where A is an agent according to the definition below.

Agents come in three flavours: Processes which perform transitions; abstractions, responsible for name and agent input; and concretions, responsible for name and agent output. Process terms are ranged over by P, Q, abstractions by F, concretions by C, and agents in general by A and B. To each well-formed agent term is assigned an arity $+w$ or $-w$, $w \in \{\mathtt{chan}, \mathtt{agent}\}^n$. A negative arity indicates the number and position of channel and agent arguments required for the agent to become a process term. Positive arities indicate arguments provided as outputs. The null arity is (), and by convention, $+() = () = -()$.

Processes Processes are agent terms of null arity: They neither require nor provide parameters to be able to perform (or refuse) transitions. Agent variables are (open) process terms; 0 is the terminated process; $P + Q$ is the process that can choose between transitions of P and of Q; $a.A$ is the prefix process that can perform an a-transition and evolve into A; $P \mid Q$ is the parallel composition of P and Q; $\nu a.P$ declares a new name a, local to P (but exportable to the outside world through subsequent communications); $\mathtt{if}\ a = b\ \mathtt{then}\ P\ \mathtt{else}\ Q$ is the conditional, often generalised to arbitrary boolean combinations of name equalities and inequalities; $P\backslash a$ is the blocking operator preventing synchronisation on the channel a; and if $D = F$ and F has arity $-\mathtt{chan}^n$ then $D(a_1, \ldots, a_n)$ is a process term too.

Abstractions We operate with two abstraction constructors, one for free input and one for bound output, similar to the situation in [4]. The free input abstraction has the shape $\lambda a.A$ ($\lambda x.A$) and has arity $-\mathtt{chan}w$ ($-\mathtt{agent}w$) if A is an abstraction term of arity $-w$ or, if w is empty, A is a process term. The arity of a bound input abstraction, $\nu\lambda a.A$, is calculated similarly.

Concretions Concretions have one of the forms $[a]A$, $\nu a.[a]A$, or $\nu\vec{a}[P]A$. The first instance corresponds to the output of the free channel name a, the second to the output of a local name a, and the third to the output of a process term P with local names $\vec{a}$. If A is a concretion term of arity $+w$ then $[a]A$ and $\nu a.[a]A$ both have arity $+\mathtt{chan}w$ and $\nu\vec{a}[P]A$ has arity $+\mathtt{agent}w$. If w is empty A is again a process term.

The Transition Semantics A standard π-calculus style semantics can easily be given to the above language. We assume a transition relation $P \xrightarrow{\tau} Q$, and a family of transition relations $P \xrightarrow{a} A$. A few examples suffice to highlight the important points:

- Invisible transitions arise because of communication. For communication to take place arities of the resulting abstraction/concretion pair must match. Thus, e.g. if $P_1 \xrightarrow{a} \nu\lambda b_1.F$ and $P_2 \xrightarrow{a} \nu b_2.[b_2]C$, F has arity $-w$, and C has arity $+w$, communication can take place. Then, if $F \mid C = Q'$, the invisible transition $P_1 \mid P_2 \xrightarrow{\tau} Q = \nu b_2.\{b_2/b_1\}Q'$ is enabled, where we assume variables to have been alpha-converted such that confusion does not arise. Similarly, if $P_1 \xrightarrow{a} \lambda x.F$ and $P_2 \xrightarrow{a} \nu\vec{b_2}.[P]C$ we obtain $P_1 \mid P_2 \xrightarrow{\tau} Q = \nu\vec{b_2}.\{P/x\}Q'$ where $Q' = F \mid C$.

- Similarly, for $P_1 \mid P_2$, it is possible that no communication takes place. Thus, eg. if $P_1 \xrightarrow{a} \lambda b.F$ and $F \mid P_2 = Q'$ then $P_1 \mid P_2 \xrightarrow{a} Q = \lambda b.Q'$. Observe again that α-conversion is used to avoid capture of variables.

- The remaining connectives reflect the intuitions given above. Thus, for instance, $\nu a.A$ declares a local name a in A and does not permit a-transitions to take place. That is, $\nu a.P \xrightarrow{b} A$ if and only if $a \neq b$ and $P \xrightarrow{b} A'$ and $A = \nu a.A'$.

3 Example: The Andrew Protocol

The agent `Alice` below represents one part of the Andrew protocol as a second-order process using the ideas outlined in the introduction:

$$\begin{aligned}
&\texttt{Alice} = \lambda K_{ab}\,.\,\texttt{in?}d\,.\,\texttt{xfer!}\{\texttt{data},d\}_{K_{ab}}\,.\\
&\qquad \texttt{Alice}(K_{ab}) + \texttt{AliceSw}(K_{ab})\\
&\texttt{AliceSw} = \lambda K_{ab}\,.\\
&\qquad \nu N_a\,.\,\texttt{xfer!}\{\texttt{switch},N_a\}_{K_{ab}}\,.\\
&\qquad\quad \texttt{xfer?}x\,.\,x \mid\\
&\qquad\quad (\texttt{AliceSw}(K_{ab}) + (K_{ab}\texttt{?}(t,N'_a,K'_{ab})\,.\\
&\qquad\quad \texttt{if } t = \texttt{next and } N_a = N'_a\\
&\qquad\quad \texttt{then xfer!}\{\texttt{ack},N_a\}_{K'_{ab}}\,.\\
&\qquad\qquad (\texttt{Alice}(K'_{ab}) + \texttt{AliceSw}(K_{ab}))\\
&\qquad\quad \texttt{else AliceSw}(K_{ab})))
\end{aligned}$$

The complementary agent `Bob` is left out for brevity. The definition uses some abbreviations. First

$$\begin{aligned}
c!(T_1,\ldots,T_n).A &\triangleq c.[T_1]\cdots[T_n]A\\
\{T_1,\ldots,T_n\}_K &\triangleq K!(T_1,\ldots,T_n).0
\end{aligned}$$

where $T_1,\ldots,T_n$ ranges over names and processes. Secondly we let $c\Gamma(v_1,\ldots,v_n).A$ abbreviate the sum of all terms of the shape $c.(\nu)\lambda v_1.\cdots.(\nu)\lambda v_n.A$ where the ν is optional, and requires the lambda to which it is applied to be a free name abstraction. Observe that this involves a non-deterministic commitment to a particular choice of input parameter types and may thus introduce deadlocks. This can be remedied, but as we are only interested in properties to hold for all possible computations the matter is insignificant.

Compared to the "standard" account little has been changed except that the protocols have been augmented with message tags to handle control flow, and a data transfer phase, in which input data is received along a channel `in`, encrypted and passed from `Alice` to `Bob`, and then output along `out`. As our aim is to specify and analyse properties in terms of external input-output behaviour some such modification is necessary, and in most parts it is completely uncontroversial. On two counts, however, some discussion is needed.

Free and Bound Input Our process language possesses the capability of detecting whether a given argument occurred freely or bound at the sender. On the face of it this is clearly an unreasonable assumption: What is received are bit strings and even if some tag of some sort states the nature of the argument how is this tag to be trustedΓ On the other hand we need this distinction in order to know, when a channel parameter is received along a trusted channel, whether to extend trust to this new channel or not. Our policy is simple: new channels communicated along trusted channels are themselves to be trusted. The argument of unreasonable expressiveness is countered by the examples always allowing for both free and bound input, as is the case above.

Looseness of Specification The data transfer phases of `Alice` and `Bob` consist simply of inputting a piece of data, encrypting and then transferring it over the medium, respectively receiving the encrypted package, decrypting and then outputting. In this respect the model is overspecific: it states explicitly, for instance, that old session keys are *not* corrupted. But this is too strong an assumption as many attacks use replays with old and corrupted session keys. Rather one would want to replace `Alice` by an *open* specification of the shape

$$\texttt{Alice} = \lambda K_{ab}\,.\,(F\ K_{ab})\texttt{;AliceSw}\ K_{ab}$$

where F is a free abstraction variable subject to assumptions such as

- F never reveals its first argument to the outside world,
- F never reveals secrets received along `in`, except when encrypted by K_{ab}.

4 Process Properties

Our intention is to formulate properties like secrecy and authenticity as functional and temporal properties expressing constraints on the input-output behaviour of the system under consideration. In our example the system consists of the agents `Alice` and `Bob` running in an unknown (and potentially hostile) environment Z. Z should be assumed to have access only to channels and data open to outside intruders. This includes the channel `xfer`. The initial value of K_{ab} should be regarded as trusted, as should the channels `in` and `out`. Suppose now that ϕ expresses a correctness property such as secrecy. The overall proof goal can then be formulated as a sequent of the shape

$$Z : \psi \vdash \nu K_{ab}.(\texttt{Alice}\ K_{ab} \mid \texttt{Bob}\ K_{ab}) \mid Z : \phi$$

where ψ is the assumption on Z (roughly: that Z does not know `in` and `out`).

Since the intruder Z is already "part of" the global system which is considered, the correctness property ϕ does not need to speak about process passing: If eg. secrecy is violated there will be a way for Z to reveal a secret along a name which is not `out`, resorting to encryption or other second-order communication only internally. More general properties which *do* talk about second-order communication will be needed once we arrive at the proofs, however.

Thus a suitable functional + temporal logic for our purpose will need to talk about names and their identities, properties of names and processes which are output, dependencies on names and properties of processes being received, in addition to usual safety properties. Observe, however, that to express the correctness properties we have in mind there is no use for liveness properties or existential path quantification. This fact will be quite useful once we come to consider the semantics. The logic has the following primitives:

- $a = b$, $a \neq b$, $\phi \wedge \psi$, $\phi \vee \psi$, $\forall a.\phi$, $\exists a.\phi$. This is just first-order logic with equality. We also need basic operations on finite sets $\vec{a}$: set membership and quantification over finite sets.
- $\vec{a}$ `fresh`, `new` $\vec{a}.\phi$. The first primitive expresses that no element of the set $\vec{a}$ occurs freely in the agent being predicated. The second primitive expresses of an agent A that it is identical to an agent of the shape $\nu\vec{b}.A'$ such that A' has the property $\{\vec{b}/\vec{a}\}\phi$. For now we can use the term "identical" as meaning, roughly, "bisimulation equivalent". We return to this issue shortly.
- $[a]\phi$, $[\tau]\phi$. These are the universal next-state quantifiers well-known from modal logic. So $[a]\phi$ will hold of an agent just in case it is a process, and whatever agent results from the performance of an a-transition must satisfy ϕ.
- $a \to \phi$, $a \to_\nu \phi$, $a \leftarrow \phi$. These primitives express name input-output properties. The first expresses of an agent A that it is an abstraction $\lambda a'.A'$, and that $\{a/a'\}A'$ has the property ϕ. The second expresses that A is an abstraction $\nu\lambda a'.A'$, and that $\{a''/a'\}A'$ has the property $\{a''/a\}\phi$ whenever a'' does not occur freely in neither $\nu\lambda a'.A$ nor ϕ (minus a). The third expresses the property that A is a concretion of the shape $[a']A'$, that $a = a'$, and that A' has the property ϕ. A fourth connective $a \leftarrow_\nu \phi$ will be derivable, as `new` $a.a \leftarrow \phi$.
- $\phi \to \psi$, $(\phi \to \psi) \to \gamma$. These primitives are used for second-order communication. The function arrow $\phi \to \psi$ expresses of A that it is identical to a second-order abstraction $\lambda x.A'$, and that if P is a process satisfying ϕ, then $\{P/x\}A'$ will have property ψ. The second primitive is a contextual property. It holds of a second-order (process) concretion A of the shape $\nu\vec{a}.[P]A'$ provided that for any receiving context f with the property $\phi \to \psi$, the process $\nu\vec{a}.(fP) \mid A'$ will have the property γ. This idea of using a second-order implication to capture contextual properties of process output originates with the paper [3].

In addition to these primitives our intention is to allow properties to be defined by greatest fixed points in the style familiar from the modal μ-calculus (cf. [7] for an adaptation to the π-calculus). This is quite straightforward if we can define the required properties using covariant recursion only. Unfortunately as yet we only have solutions that make use of contravariant recursion, and thus we need to address the foundational problem of making sense of this. This we do in the subsequent sections. First, however, some syntactical matters: Recursively defined properties take the shape $(\nu X(a_1, \ldots, a_n).\phi)(b_1, \ldots, b_n)$ (cf. [4]). Alternatively we use the sugared form $X(b_1, \ldots, b_n)$ in the context of a definition of the shape $X(a_1, \ldots, a_n) => \phi$. We require that recursive definitions are *guarded* in the sense that all occurrences of X in ϕ must be within the scope of either a modal operator, or one of the arrows. A formula ϕ is *propositionally closed* if ϕ does

not have free occurrences of (parametrised) variables X.

5 Expressing Trust

The following predicate expresses a property of trust in a finite set $\vec{a}$ of channels. The example is given for monadic communication only. The generalisation to polyadic communication is quite easy and can be found in the full version of the paper.

$\vec{a}$ `trusted` =>
$[\tau](\vec{a}$ `trusted` $\wedge)$
$\forall b.[b](\vec{a}$ `trusted` $\vee$
$\vec{a}$ `trusted_out_after` $b \vee$
$\vec{a}$ `trusted_in_after` $b)$

$\vec{a}$ `trusted_out_after` b =>
$(\exists c.c \leftarrow (\vec{a}$ `trusted` $\wedge (c \in \vec{a} \supset b \in \vec{a}))) \vee$
(`new` $c.c \leftarrow$ $((b \in \vec{a} \supset \vec{a}, c$ `trusted`$) \wedge$
$(b \notin \vec{a} \supset \vec{a}$ `trusted`$))) \vee$
$(\forall \vec{d}.\vec{d}$ `fresh` $\supset$ `new` $\vec{c}.$
$(\vec{a}, \vec{d}, \vec{c}$ `trusted` $\rightarrow \vec{a}, \vec{d}, \vec{c}$ `trusted`$)$
$\rightarrow \vec{a}, \vec{d}, \vec{c}$ `trusted`$)$

$\vec{a}$ `trusted_in_after` b =>
$(\forall c.c \rightarrow \vec{a}$ `trusted`$) \vee$
$(c \rightarrow_\nu (b \in \vec{a} \supset \vec{a}, c$ `trusted`$) \wedge$
$(b \notin \vec{a} \supset \vec{a}$ `trusted`$)) \vee$
$(\forall \vec{c}.\vec{c}$ `fresh` $\supset \vec{a}, \vec{c}$ `trusted` $\rightarrow \vec{a}, \vec{c}$ `trusted`$))$

The idea is quite simple: To show that the process being predicated respects the trustedness of names in $\vec{a}$ we need to consider the various transitions that may be possible from the initial state and the various types of continuation agents that may ensue. For instance, free outputs must be trusted only if the synchronisation channel is, and the continuation must preserve trust as stated. Bound outputs along trusted channels cause trust to be extended. For second order input the process being input must respect the trustedness of $\vec{a}$, evidently. But in addition we must permit that process to mention other trusted information of which we are not yet aware. This information will be "fresh" to us, and we had better ensure that after input of the process we respect the trustnedness of both $\vec{a}$ and $\vec{c}$ (as it were).

Observe the two contravariant occurrences of the trustedness predicate, for the cases of second order input and output. We see no possibility at present of avoiding these. Freeness checks, for instance, are clearly much too inexpressive. On the other hand the semantics of the modal μ-calculus on which the logic is built rests on the fact that fixed points are required to be computed of monotone operations only, and in the presence of contravariant recursion monotonicity will fail. This issue is addressed extensively in the full version of the paper where we give two alternative semantics of the logic, one based on intervals and standard fixed point semantics, and an iterative semantics. Here we present only the latter.

6 An Iterative Semantics

Our intention is to compute the semantics of a formula ϕ as the limit of an increasing chain of sets of agents $\|\phi\|^n(\sigma)$, where $n \in \omega$ is an *approximation index* and σ is an interpretation of predicate variables as sets of agents (for simplicity we consider only set variables). At each iteration step, $\|\phi\|^n(\sigma)$ will be a set of agents which is permitted to depend on the behavior of agents only down to a global transition depth n. To get at this notion we introduce a version of the simulation preorder.

Definition 1 (Simulation Preorder)

1. Define the preorders $\preceq_n$ inductively by the following clauses (where we use f to range over functions from names to abstractions or processes to abstraction, as appropriate given the context):
   (a) $P \preceq_0 Q$ holds always.
   (b) $P \preceq_{n+1} Q$ iff $fn(P) = fn(Q)$ and $Q \xrightarrow{\alpha} B$ implies $P \xrightarrow{\alpha} A$ such that $A \preceq_n B$.
   (c) $\lambda x.A_1 \preceq_{n+1} \lambda y.A_2$ iff for all a (P), $\{a/x\}A_1 \preceq_n \{a/y\}A_2$ $(\{P/x\}A_1 \preceq_n \{P/y\}A_2)$.
   (d) $[a]A_1 \preceq_{n+1} [b]A_2$ iff $a = b$ and $A_1 \preceq_n A_2$, $\nu a.[a]A_1 \preceq_{n+1} \nu b.[b]A_2$ iff for all fresh c, $\{c/a\}A_1 \preceq_n \{c/b\}A_2$, $\nu\vec{a}.[P]A \preceq_{n+1} \nu\vec{b}.[Q]B$ iff for all process abstractions $\lambda x.A'$ for which $\vec{a}$ and $\vec{b}$ are fresh, $\nu\vec{a}.\{P/x\}A' \mid A \preceq_n \nu\vec{b}.\{Q/x\}A' \mid B$.
2. $A \preceq B$ iff for all $n \in \omega$, $A \preceq_n B$. $A \approx B$ iff $A \preceq B$ and $B \preceq A$.
3. Let S be a set of agents. Then $\uparrow_n S = \{B \mid \exists A \in S.A \preceq_n B\}$.

Observe that $\approx$, being the intersection of a simulation order and its converse, is strictly coarser than bisimulation equivalence. We can now introduce the semantics:

$\|\phi\|^0 = \{A \mid A \text{ an agent}\}$
$\|X\|^{n+1}(\sigma) = \uparrow_{n+1} \sigma(X)$
$\|\nu X.\phi\|^{n+1}(\sigma) = \|\phi\|^{n+1}(\{\|\nu X.\phi\|^n(\sigma)/X\}\sigma)$
$\|\phi \wedge \psi\|^{n+1}(\sigma) = \|\phi\|^{n+1}(\sigma) \cap \|\psi\|^{n+1}(\sigma)$

$$\|\phi \vee \psi\|^{n+1}(\sigma) = \|\phi\|^{n+1}(\sigma) \cup \|\psi\|^{n+1}(\sigma)$$
$$\|\vec{a}\ \texttt{fresh}\|^{n+1}(\sigma) = \{A \mid \vec{a} \cap fn(A) = \emptyset\}$$
$$\|\texttt{new}\ \phi\|^{n+1}(\sigma) = \{A \mid \nu\vec{a}.A' \preceq_{n+1} A, A' \in \|\phi\|^{n+1}(\sigma)\}$$
$$\|[\alpha]\phi\|^{n+1}(\sigma) = \{P \mid P \xrightarrow{\alpha} A \supset A \in \|\phi\|^{n}(\sigma)\}$$
$$\|a \to \phi\|^{n+1}(\sigma) = \{\lambda b.A \mid \{a/b\}A \in \|\phi\|^{n}(\sigma)\}$$
$$\|a \to_\nu \phi\|^{n+1}(\sigma) = \{\lambda b.A \mid \{a'/b\}A \in \|\{a'/a\}\phi\|^{n}(\sigma), \quad a \notin fn(\lambda b.A) \cup (fn(\phi) - \{a\})\}$$
$$\|a \leftarrow \phi\|^{n+1}(\sigma) = \{[a]A \mid A \in \|\phi\|^{n}(\sigma)\}$$
$$\|\phi \to \psi\|^{n+1}(\sigma) = \|\phi\|^{n}(\sigma) \to \|\psi\|^{n}(\sigma)$$
$$\|(\phi \to \psi) \to \gamma\|^{n+1}(\sigma) = \{\nu\vec{a}.[P]A \mid \forall(\lambda x.A') \in \|\phi\|^{n}(\sigma) \to \|\psi\|^{n}(\sigma). \ \nu\vec{a}.\{P/x\}A' \mid A \in \|\gamma\|^{n}(\sigma)\}$$

Observe that guardedness is important for this definition to make sense. Abbreviate $\|\phi\|^n(\sigma)$ by $\|\phi\|^n$ when ϕ is propositionally closed, and let $\|\phi\| = \bigcap_{n\in\omega} \|\phi\|^n$.

7 The Andrew Protocol: Specification

We adopt the following intuitive account of secrecy:

- *Secrecy:* A fresh piece of datum (ie. a secret) received along `in` can only be output along a secret channel.

Our aim is to formalise this as a formula ϕ for which the following kind of sequent should be established

(1) $Z : \{\texttt{in}, \texttt{out}\}\ \texttt{fresh} \vdash (\nu K_{ab}.\texttt{Alice}\ K_{ab} \mid \texttt{Bob}\ K_{ab}) \mid Z : \phi.$

Finding such a ϕ is not difficult:

$\vec{a}$ `secret =>`
$[\tau](\vec{a}\ \texttt{secret}) \wedge$
$[\texttt{in}](\texttt{bound\_input} \supset b \to_\nu \vec{a}, b\ \texttt{secret})$
$\forall c.[c](\texttt{free\_output} \supset \exists d.d \leftarrow ((d \in \vec{a} \supset c \in \vec{a}) \wedge \vec{a}\ \texttt{secret}))$

We use the following two ancillary predicates:

$\texttt{bound\_input} = a \to_\nu \ \texttt{true}$
$\texttt{free\_output} = \exists a.a \leftarrow \ \texttt{true}$

The property ϕ of (1) becomes $\{\texttt{in}, \texttt{out}\}\ \texttt{secret}$. The specification of secrecy reflects the intuition very closely. Secrets are either members of the initial value of $\vec{a}$, or they have sometime been input along `in` as a fresh name. Observe that only traces of τ-transitions, name inputs along `in`, or free outputs are considered. This is admissible as correctness is stated of an *open* system: If we accidentally choose a Z which violates secrecy by, say, passing secret-revealing processes to the outside world through an unsafe channel, then there will be another Z which decodes these secret-revealing processes to pass out the (first-order) secrets in a manner that will violate the proof goal.

8 Proving Trust

Secrecy is proved using the `trusted` predicate introduced earlier. Proofs can be found in the full version of the paper.

Lemma 2 $X : \vec{a}, \texttt{in trusted} \vdash X : \vec{a}, \texttt{in secret}$

The problem of proving secrecy is thus "reduced" to the problem of proving trust. The point of the trustedness predicate is that it lends itself to a structural analysis. The verification takes the shape of series of lemmas intended to support this structural analysis. The most difficult issue is how to deal with parallel composition. In this case we need to be careful about the creation of new internal resources. We extend the sequent notation slightly, following the suggestion of [4], by writing, eg., $X : \phi, Y : \psi \vdash^{\vec{b}} X \mid Y : \gamma$ to express that whenever X satisfies ϕ and Y satisfies ψ, then $\nu\vec{b}.(X \mid Y)$ satisfies γ, where the scope of $\vec{b}$ includes both ϕ and ψ (but not γ). The delicate part of the trustedness predicate is to deal with the situations in which the "coverage" of the trust predicate needs to be modified because trusted channels are given local scopes, or because trust needs to be extended to channels that are currently unknown to the agent being predicated.

For the proofs we need lemmas of the following sort:

Lemma 3

1. $X : \vec{a}\ \texttt{trusted}, X : \vec{b}\ \texttt{fresh} \vdash X : \vec{a}, \vec{b}\ \texttt{trusted}$
2. *For all* P, $\vdash P : \emptyset\ \texttt{trusted}$

A consequence of lemma 3 is that $P : \vec{a}\ \texttt{trusted}$ for any set $\vec{a}$ of names that do not occur in P. The next lemma shows that trusted names can safely be localised.

Lemma 4 $X : \vec{a}, \vec{b}\ \texttt{trusted} \vdash^{\vec{b}} X : \vec{a}\ \texttt{trusted}$

For the case of process outputs we also need to consider expanding the set of trusted names to fresh ones for functions:

Lemma 5 *Assume that* $X : \vec{a}\ \texttt{trusted} \vdash A : \vec{a}\ \texttt{trusted}$ *and* $\vec{b} \cap \mathrm{fn}(A) = \emptyset$. *Assume also that* $\vdash B : \vec{a}, \vec{b}\ \texttt{trusted}$. *Then* $\vdash \{B/X\}A : \vec{a}, \vec{b}\ \texttt{trusted}$.

The proof of this lemma turns out to be surprisingly delicate, and requires techniques that are somewhat different from the quite elementary techniques used elsewhere in this section. Essentially lemma 5 states a property which is much more "intensional" than the corresponding property 3, concerning, as it does,

function definability: All functions in $\vec{a}$ `trusted` $\rightarrow$ $\vec{a}$ `trusted` that do not "mention" $\vec{b}$ can be extended to functions in $\vec{a},\vec{b}$ `trusted` $\rightarrow \vec{a},\vec{b}$ `trusted`.

Using the stated lemmas we can then proceed to the first main result, showing that the trustedness predicate is preserved by parallel composition:

Lemma 6
$X : \vec{a},\vec{b}$ `trusted`$, Y : \vec{a},\vec{b}$ `trusted` $\vdash^{\vec{b}} X \mid Y : \vec{a}$ `trusted`

9 Deriving a Type System

In this section we show how a type system for inferring judgments of the form $\Gamma \vdash^{\vec{a}} A : \vec{b}$ `trusted` can be derived from the results achieved so far. Here Γ is a set of hypotheses which are either boolean combinations of name equations or inequations, or of one of the forms $x : \vec{a}$ `trusted`, or $f : \vec{a}$ `trusted` $\rightarrow \vec{a}$ `trusted`. The proof system uses an ancillary relation $\Gamma \vdash^{\vec{b}} A : c$ `fresh` to hold if c does not occur freely in $\nu\vec{b}.A$, and whenever x (f) is a process (function) variable occurring in Γ then $\Gamma \vdash x : c$ `fresh`. For concerns of space we leave out most rules from the abstract and give just a few examples.

The inference system include structural rules to preform case analysis, and to reflect most lemmas of the previous section, such as:

$$\frac{\Gamma \vdash^{\vec{b}} P : \vec{a},\vec{c}\ \texttt{trusted}}{\Gamma \vdash^{\vec{b},\vec{c}} P : \vec{a}\ \texttt{trusted}}$$

$$\frac{\Gamma \vdash^{\vec{b}} P : \vec{a}\ \texttt{trusted} \quad \Gamma \vdash^{\vec{b}} P : \vec{c}\ \texttt{fresh}}{\Gamma \vdash^{\vec{b}} P : \vec{a},\vec{c}\ \texttt{trusted}}$$

$$\frac{\Gamma \vdash A : \vec{a}\ \texttt{trusted} \rightarrow \vec{a}\ \texttt{trusted} \quad \Gamma \vdash A : \vec{c}\ \texttt{fresh}}{\Gamma \vdash A : \vec{a},\vec{c}\ \texttt{trusted} \rightarrow \vec{a},\vec{c}\ \texttt{trusted}}$$

Secondly a set of term rules are needed to structurally decompose trust assertions, such as:

$$\frac{\cdot}{\Gamma \vdash^{\vec{b}} 0 : \vec{a}\ \texttt{trusted}}$$

$$\frac{\Gamma \vdash P : \vec{a}\ \texttt{trusted} \quad \Gamma \vdash Q : \vec{a}\ \texttt{trusted}}{\Gamma \vdash P \mid Q : \vec{a}\ \texttt{trusted}}$$

$$\frac{\Gamma' \vdash^{\vec{b},\vec{c}} A : \vec{a}\ \texttt{trusted}}{\Gamma \vdash^{\vec{b}} \nu\vec{c}.A : \vec{a}\ \texttt{trusted}} \quad \text{(See below)}$$

$$\frac{\Gamma \vdash^{\vec{b}} A : \vec{a}\ \texttt{trusted} \quad c\ \textit{fresh}}{\Gamma \vdash^{\vec{b}} \lambda c.A : \vec{a}\ \texttt{trusted\_in\_after}\ d}$$

$$\frac{\Gamma \models d \in \vec{a} \quad c\ \textit{fresh} \quad \Gamma \vdash^{\vec{b}} A : \vec{a},c\ \texttt{trusted}}{\Gamma \vdash^{\vec{b}} \nu\lambda c.A : \vec{a}\ \texttt{trusted\_in\_after}\ d}$$

The set Γ' in rule Nu is computed in the following way:

$$\begin{aligned}\Gamma' \quad = \quad & \Gamma \cup \{x : \vec{c}\ \texttt{fresh} \mid x \text{ mentioned in } \Gamma\} \\ & \cup \{f : \vec{c}\ \texttt{fresh} \mid f \text{ mentioned in } \Gamma\}\end{aligned}$$

To terminate proof construction we have the following rule of unfolding and discharge:

$$\frac{\begin{array}{c}[\Gamma' \vdash^{\vec{b'}} D(d_1,\ldots,d_n) : \vec{a'}\ \texttt{trusted}] \\ \vdots \\ \Gamma \vdash^{\vec{b}} F(c_1,\ldots,c_n) : \vec{a}\ \texttt{trusted}\end{array}}{\Gamma \vdash^{\vec{b}} D(c_1,\ldots,c_n) : \vec{a}\ \texttt{trusted}}$$

The rule is subject to the sidecondition that $\Gamma' \vdash^{\vec{b'}} D(d_1,\ldots,d_n) : \vec{a'}$ `trusted` is a substitution instance of $\Gamma \vdash^{\vec{b}} D(c_1,\ldots,c_n) : \vec{a}$ `trusted`, and that the assumed deduction

$$\frac{\begin{array}{c}\Gamma' \vdash^{\vec{b'}} D(d_1,\ldots,d_n) : \vec{a'}\ \texttt{trusted} \\ \vdots \\ \Gamma \vdash^{\vec{b}} F(c_1,\ldots,c_n) : \vec{a}\ \texttt{trusted}\end{array}}{}$$

is non-trivial in the sense that it includes the application of a term rule (cf. similar side conditions in [7]). The type system is sound. We conjecture that the system is complete and decidable for the fragment without blocking. The full version of the paper contains a proof of secrecy of the Andrew protocol using the type system.

10 Conclusion

We have shown how to represent key management protocols as second-order processes, how to specify secrecy as a higher-order temporal logic formula, how to give semantics to such a logic in face of contravariant recursion, and how to prove properties of trust in structural terms.

As an approach to analysing correctness properties of security and authentication protocols our approach suffers serious shortcomings which we have not yet resolved. First, trust in monotonically increasing sets of channels is not of huge practical interest. Protocols, and protocol users, in particular, must be permitted to revoke trust in resources such as old session keys that are no longer in use. However, our trust predicate does not permit this, as it does not reflect the protocol features that govern trust revocation (eg. trust in the current session key can be revoked once a new session key has been agreed upon). Observe though that this problem is shared with other approaches in the literature to information flow analysis based on type

inference or static analysis. In future work we will have to investigate more refined versions of the trust predicate. Also we have not yet considered proofs of authenticity. Another important aspect is to represent other encryption primitives such as public-key encryption and computed keys. Structured channels as in [10] appear useful and could easily be accomodated. Finally we need to address the correctness of our representation of encryption, and to what extent the higher-order model is really necessary and useful. An alternative could be to reduce to π-calculus proper as suggested in the full version of [2]. This option and its relation to our higher-order model would be worth exploring further. A noteworthy point is that in the presence of the blocking operator the reduction of higher-order processes to first-order ones by Sangiorgi [12] is not applicable. We are currently accumulating strong evidence to suggest that the reduction from the second-order calculus with blocking to the first-order π-calculus with or without blocking while feasible in principle is very complicated and definitely not suitable as a modelling tool.

Acknowledgements

Thanks are due to Martín Abadi, José Luis Vivas, Alan Mycroft and Dilian Gurov for comments and discussions on several topics treated here. It is the credit of Jose-Luis to have observed the need for firewalling using the blocking operator. Also thanks are due to one anonymous referee in particular for some very insightful comments. The work was partially supported by Esprit BRA 8130 LOMAPS and a Swedish Foundation for Strategic Research Junior Individual Grant. Part of the work was done while visiting CMI, Université d'Aix Marseille 1. The full version is available as `ftp://ftp.sics.se/pub/fdt/mfd/ptssop.ps.Z`.

References

[1] M. Abadi. Secrecy by typing in security protocols (draft). Manuscript, Available through http://www.research.digital.com/SRC/, 1997.

[2] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Proc. 4th ACM Conference on Computer and Communications Security*, pages 36–47, 1997. Full version available as tech. rep. 414, Univ. Cambridge Computer Lab.

[3] R. Amadio and M. Dam. Reasoning about higher-order processes. In *Proc. CAAP'95,* Lecture Notes in Computer Science, 915:202–217, 1995.

[4] R. Amadio and M. Dam. A modal theory of types for the π-calculus. In *Proc. FTRTFT'96,* Lecture Notes in Computer Science, 1135:347–365, 1996.

[5] J.-P. Banâtre, C. Bryce, and D. Le Metayer. Compile time detection of information flow in sequential programs. In *Proc. European Symp. on Research in Computer Security,* LNCS 875, pages 55–73, 1994.

[6] M. Burrows, M. Abadi, and R. M. Needham. A logic of authentication. *Proc. Royal Society of London A*, 1989.

[7] M. Dam. Model checking mobile processes. *Information and Computation*, 129:35–51, 1996.

[8] D. Denning. Certification of programs for secure information flow. *Communications of the ACM*, 20:504–513, 1977.

[9] R. Focardi and R. Gorrieri. The compositional security checker: A tool for the verification of information flow properties. To appear in *IEEE Transactions on Software Engineering.*

[10] G. Lowe. Breaking and fixing the Needham-Schroeder public-key authentication protocol. *Proc. TACAS,* Lecture Notes in Computer Science, **1055**:147–166, 1996.

[11] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I and II. *Information and Computation*, 100(1):1–40 and 41–77, 1992.

[12] Davide Sangiorgi. From π-calculus to Higher-Order π-calculus — and back. in *Proc. TAPSOFT'93* Lecture Notes in Computer Science, 668:151–166, 1993.

[13] B. Thomsen. A calculus of higher order communicating systems. In *Proc. POPL'89*, pages 143–154, 1989.

[14] D. Volpano, G. Smith, and C. Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 4:1–21, 1996.