

A Calculus for Trust Management

Marco Carbone,¹ Mogens Nielsen,¹ Vladimiro Sassone²

¹ BRICS*, University of Aarhus ² Dept. of Informatics, University of Sussex

Abstract. We introduce *ctm*, a process calculus which embodies a notion of trust for global computing systems. In *ctm* each principal (location) is equipped with a policy, which determines its legal behaviour, and with a protocol, which allows interactions between principals and the flow of information from principals to policies. We elect to formalise policies using a Datalog-like logic, and to express protocols in the process algebra style. This yields an expressive calculus very suitable for the global computing scenarios, and provides a formalisation of notions such as trust evolution. For *ctm* we define barbed equivalences and study their possible applications.

1 Introduction

In the last few years Global Computing (GC) has emerged as an important part of computer science. A GC system is composed of entities which are autonomous, decentralised, mobile, dynamically configurable, and capable of operating under partial information. Such systems, e.g. the Internet, become easily very complex, and bring forward the need to guarantee security properties. Traditional security mechanisms, however, have severe limitations in this setting, as they often are either too weak to safeguard against actual risks, or too stringent, imposing unacceptable burdens on the effectiveness and flexibility of the infrastructure. Trust management systems, in which safety critical decision are made based on trust policies and their deployment in the presence of partial knowledge, have been proposed as an alternative attempting to deal with those limitations.

Building on the experience of our previous work on trust [8] we introduce a process calculus for modelling trust management systems (*ctm*). Many models for trust based system have appeared in the literature, and most of them feature some sort of logic to describe trust policies. However, lacking a notion of protocol, such approaches typically fall short to describe the exact behaviour of systems, which is a fundamental property when security concerns are present. Consider for instance a server whose policy is to grant access only to a certain class of users, but whose (flawed) protocol of communication always allows access to a particular resource. Even though the policy may be correct, the whole system is not. A second aspect of paramount importance

Marco Carbone and Mogens Nielsen supported by ‘**SECURE**: Secure Environments for Collaboration among Ubiquitous Roaming Entities’, EU FET-GC IST-2001-32486. Marco Carbone supported by ‘**DisCo**: Semantic Foundations of Distributed Computation’, EU IHP ‘Marie Curie’ HPMT-CT-2001-00290. Vladimiro Sassone supported by ‘**MyThS**: Models and Types for Security in Mobile Distributed Systems’, EU FET-GC IST-2001-32617. *Basic Research In Computer Science funded by the Danish National Research Foundation.

here is to allow principals' interactions to feedback to the security mechanisms and influence future policies. For instance, access rights can change for a principal in response to its behaviour, and how precisely this behaviour should be observed and the variation should take place ought to be part of the model.

The aim of this work is to develop a coherent framework centred on these two aspects, and establish its basic theory. In `ctm`, a principal is specified by a pair, a policy α and a protocol P , which interact in precise ways, as described below. The policy α informs the protocol P as to what actions are allowed at any given time, and works on the basis of evidence and observations collected from past interactions. Dually, P interacts with a network of other principals, and in doing so it produces the observations gathered in α . The protocol P will consult α when making a decision, e.g. whether or not to grant a specific service to a specific principal. Schematically, we can represent the situation as in the informal picture below.

$$(\text{Policy} \iff \text{Protocol}) \parallel \text{Network}$$

We model the ‘policy’ side of the drawing with a decidable logic. The choice is a Datalog-like logic: a principal’s policy will be represented as a set of formulas depending on a set of past observations. On the ‘protocol’ side, our model is based on a process calculus in the style of the π -calculus [14]. More precisely, `ctm` is a calculus with locations linked by channel names. Each location (uniquely) identifies a principal, and the diagram above would be represented as $a\{P\}_\alpha \mid N$, where a is a principal with protocol P and policy α , in parallel with the rest of the network N . In `ctm` we associate the action of sending a message to another principal as granting a particular resource (viz. the service represented by the channel name). Outputs will then be guarded by a formula ϕ from the logic, as for instance $\phi :: b \cdot \ell\langle\tilde{m}\rangle$, which according to the truth value of ϕ allows the protocol to send \tilde{m} to b on channel (or method) ℓ . As a concrete example, a protocol like $\text{Access}(b, R) :: b \cdot l\langle n \rangle$ would stand for ‘if my policy grants b ‘Access’ to R , then send n along l to b .’ Symmetrically, inputs represent requests of services, and form the observable basis mentioned above. For instance, if executing an input action $b \cdot \text{print}(y).P$, we receive a message ‘junk’ for y from b , we observe and record that b has attempted to print a corrupted file. As mentioned above, multiple channels at a single location allows to distinguish among different services provided by a principal. We assume in `ctm` that the identity of communicating principals cannot be corrupted (i.e. we assume implicitly that authenticity etc. is guaranteed by lower level protocols).

In order to allow principals to offer services to generic (as opposed to named) clients, `ctm` features a new form of input capability, which allows to abstract from the communicating principal. For instance, in order to offer a printing service for all, we would write $x \cdot \text{print}(y).P$, where x is a variable, which at the time of interaction will be bound to the name of the principal requesting the service. We call this operation *global input*.

The calculus `ctm` seems a powerful tool for expressing several examples of trust-based systems for GC. Casting these ingredients in the world of process algebras provides many interesting theoretical notions, and in particular behavioural equivalences. The natural separation of principals in pairs of components, viz. policies and protocols, induces new notions of equivalences. In particular, besides studying equivalences of

principals and networks, one can focus on protocols as well as policies. Technically, the main contribution of this paper is to introduce a theory of observational equivalence for trust-based systems, which captures in a single, homogeneous framework with equivalence of protocols, policies, and principals.

Related Work. To the best of our knowledge, the notion of trust has never been fully treated in process calculi. In $D\pi$ [3,10] policies are statically specified, not allowing dynamic updates; [9] considers a formalism for cryptographic protocols, similar to ours: communications are guarded by logical formulas meant for proving correctness, whereas protocols are expressed with strand-spaces. Concerning policies for access control, there are many works on logics, where a trust engine is responsible for constructing [6,11,12] or checking [4] a proof that a desired request is valid. In [13] and [5] authors provide a decidable logic for policies, proposing variants of Datalog. In particular, *Cassandra*, provides a formalism for expressing policies in a GC scenario, where, as in our case, each principal has its own policy. They also allow references to other principals' policies and delegation, using fixed-point computations as in [8].

Plan of the paper. Section 2 defines the calculus: logic for policies, syntax and semantics of networks and protocols. In Section 3 we study barbed equivalences on protocols, policies and principals, while Section 4 is about the expressiveness of global input.

2 The Calculus

Let Val be a denumerable set of values ranged over by l, m and partitioned into sets P and N , respectively the set of principals (ranged over by a, b, c) and the set of names (ranged over by n). Moreover Var (ranged over by x, y, z) is a set of variables such that $\text{Var} \cap \text{Val} = \emptyset$. In the sequel we assume u, v in $\text{Var} \cup \text{Val}$ and p in $\text{Var} \cup \text{P}$. As usual a tilde over a letter indicates the extension to vectors.

A small logic for policies. As explained in the introduction, each principal acts on a body of knowledge built on the past interactions with other principals. We represent such information using the notion of interaction datatype. Messages in our calculus have form $a \cdot \tilde{l} \triangleright \tilde{m}$ representing a message \tilde{m} from principal a on channel \tilde{l} .

Definition 1 (Interaction Datatype). An interaction datatype \mathcal{M} over Val is a triple $(\mathcal{S}, \mathcal{R}, \text{upd})$ where \mathcal{S} is a generic set of so-called interaction values, \mathcal{R} is a set of decidable subset of $\mathcal{S} \times \text{Val}^k$, and upd is a function which given $s \in \mathcal{S}$ and a message $a \cdot \tilde{l} \triangleright \tilde{m}$ returns an element of \mathcal{S} .

According to the above definition, the set \mathcal{S} is a generic set: the idea is to build elements of \mathcal{S} as representation of abstract information about past interactions with other principals. The set \mathcal{R} defines the basic predicates binding together interaction values and elements of Val , and upd defines the effect in \mathcal{S} by receiving a message.

Example 1 (Lists and Multisets). Let \mathcal{S} be the set of lists with elements $a \cdot \tilde{l} \triangleright \tilde{m}$, i.e. $\mathcal{S} = \{[a_1 \cdot \tilde{l}_1 \triangleright \tilde{m}_1, \dots, a_k \cdot \tilde{l}_k \triangleright \tilde{m}_k] \mid k \geq 0\}$ and upd the operation of list concatenation. The set \mathcal{R} could contain the relation $\text{last}_{\tilde{m}}$ which holds true of lists whose last element carries the message \tilde{m} , and the relation $\text{from}_{\geq 5}(a)$, satisfied whenever the number of

messages in the list from a is larger than 5. Another interesting example is when \mathcal{S} is the set of multisets over elements $a \cdot \tilde{l} \triangleright \tilde{m}$ with multiset union as upd. Predicates can express the number of message occurrences, e.g. predicate $x \cdot - \triangleright y < k$ is satisfied by all elements of \mathcal{S} such that the number of occurrences of elements $x \cdot z \triangleright y$ is less than k .

Principals use policies to make decisions based on the information contained in an element $s \in \mathcal{S}$ of a given interaction datatype \mathcal{M} .

Definition 2 (Policy). Let $\mathcal{M} = (\mathcal{S}, \mathcal{R}, \text{upd})$ be an interaction datatype, let \mathcal{P} and $\mathcal{P}_{\mathcal{M}}$ be disjoint signatures of predicates symbols, with $\mathcal{P}_{\mathcal{M}}$ in one-to-one correspondence with \mathcal{R} . A policy π is defined as a set of rules of type $L(\tilde{u}) \leftarrow L_1(\tilde{u}_1), \dots, L_k(\tilde{u}_k)$ such that $L \in \mathcal{P}$ and $L_i \in \mathcal{P} \cup \mathcal{P}_{\mathcal{M}}$.

π is interpreted as a Datalog program [2] relative to an element $s \in \mathcal{S}$. More precisely, each rule in π is interpreted as Datalog implication, where predicate symbols in $\mathcal{P}_{\mathcal{R}}$ take as an implicit first argument the interaction value s . Given a pair (π, s) and a predicate $A(\tilde{l})$ we write $(\pi, s) \vdash A(\tilde{l})$ meaning that $A(\tilde{l})$ is entailed by the Datalog program π relative to s . In the sequel, letters α, β will denote pairs (π, s) .

Syntax for the calculus. Let \mathcal{M} be a interaction datatype, \mathcal{P} a signature. The syntax of ctm is then featured by two main syntactic categories: *networks* (N) and *protocols* (P, Q).

$N, M ::= \epsilon$	(empty)	$P, Q ::= \mathbf{0}$	(null)
$ N N$	(net-par)	$ Z$	(sub)
$ a\{ P \}_\alpha$	(principal)	$ P P$	(par)
$ (\nu n) N$	(new-net)	$ (\nu n) P$	(new)
		$!P$	(bang)
$Z ::= p \cdot \tilde{u}(\tilde{v}) \cdot P$	(input)		
$ \phi :: p \cdot \tilde{u}(\tilde{v}) \cdot P$	(output)	$\phi ::= L(\tilde{l}) \quad L \in \mathcal{P}$	(null)
$ Z + Z$	(sum)	$\phi \wedge \phi$	(and)

A network N is composed of principals running in parallel. Each principal is equipped with a protocol P and a policy α . From now on we assume to work only with networks N where principals names are unique, i.e. for each $a \in \mathbf{P}$ there is at most one subterm of N of the kind $a\{ P \}_\alpha$.

A protocol P is given in the style of π -calculus [14]. The protocol $\mathbf{0}$ represents the inactive process. Terms (input) and (output) represent the main actions, and both can be part of the standard sum operator (guarded choice). As remarked in the introduction, the input capability can either refer to a specific principal, or be global. The output action sends a message on a channel and is guarded by a predicate ϕ in the signature \mathcal{P} . For generality, we allow composite channel names as in [7]. The (bang) and (par) operators are standard. The rest of the paper will omit trailing inactive processes.

The set of free names fn (resp. bound names bn) and free variables fv (resp. bound variables bv) are defined as usual on networks and protocols. Closed and open terms are

defined as usual (with respect to variables). The symbol σ denotes a substitution from variables to names. Applying a substitution σ to a network N (or a protocol P) will be denoted by $N\sigma$ ($P\sigma$). The global input variable is a strong binder, e.g. in $x \cdot l(y).x \cdot l(y)$ the first x binds the second, instead the first y does not bind the second y . We omit trivial guards from outputs, i.e. $\tau\tau :: b \cdot \tilde{l}(\tilde{m})$ will be written as $b \cdot \tilde{l}(\tilde{m})$ where $\tau\tau$ denotes the “always” true predicate.

Reduction Semantics. In this section we give the formal semantics of the calculus in terms of reduction semantics. The structural congruence relation \equiv is the least congruence relation on N such that $|$ and $+$ are commutative monoids on protocols, $|$ is a commutative monoid on networks, and such that it satisfies alpha-conversion and the rules

$$\begin{aligned}
(\text{Struct}_1) \quad & a\{ !P \mid Q \}_\alpha \equiv a\{ P \mid !P \mid Q \}_\alpha \\
(\text{Struct}_2) \quad & (vn) (vn') W \equiv (vn') (vn) W \text{ for } W \in \{P, N\} \\
(\text{Struct}_3) \quad & a\{ (vn) P \mid Q \}_\alpha \equiv a\{ (vn) (P \mid Q) \}_\alpha \text{ if } n \notin fn(Q) \\
(\text{Struct}_4) \quad & (vn) N \mid M \equiv (vn) (N \mid M) \text{ if } n \notin fn(M) \\
(\text{Struct}_5) \quad & a\{ (vn) P \}_\alpha \equiv (vn) a\{ P \}_\alpha
\end{aligned}$$

We define \rightarrow as the least binary relation on N satisfying the rules given in Table 1. Rule (RCom) defines communication between two principals. For $\alpha = \langle \pi, s \rangle$, the operator $\alpha \oplus [b \cdot \tilde{l} \triangleright \tilde{m}]$ returns a new $\alpha' = \langle \pi, s' \rangle$ such that $s' = \text{upd}(s, b \cdot \tilde{l} \triangleright \tilde{m})$. The operator \odot is defined on tuples, as the most general unifier returning a substitution σ , whose application in the semantics is conditioned by successful unification. The rule (RINT) describes internal communication and is similar to (RCom). Rules (RSTRUCT) and (RPAR) are standard. As usual we define \rightarrow^* as the reflexive and transitive closure of \rightarrow .

$$\begin{aligned}
(\text{RCom}) \quad & \frac{\beta \vdash \phi \quad \alpha' = \alpha \oplus [b \cdot \tilde{l} \triangleright \tilde{m}] \quad b : \tilde{m} \odot p : \tilde{x} = \sigma}{a\{ p \cdot \tilde{l}(\tilde{x}), P+P' \mid P'' \}_\alpha \mid b\{ \phi :: a \cdot \tilde{l}(\tilde{m}), Q+Q' \mid Q'' \}_\beta \rightarrow a\{ P\sigma \mid P'' \}_\alpha \mid b\{ Q \mid Q'' \}_\beta} \\
(\text{RINT}) \quad & \frac{\alpha \vdash \phi \quad \alpha' = \alpha \oplus [a \cdot \tilde{l} \triangleright \tilde{m}] \quad a : \tilde{m} \odot p : \tilde{x} = \sigma}{a\{ p \cdot \tilde{l}(\tilde{x}), P+P' \mid \phi :: a \cdot \tilde{l}(\tilde{m}), Q+Q' \mid Q'' \}_\alpha \rightarrow a\{ P\sigma \mid Q \mid Q'' \}_\alpha} \\
(\text{RRES}) \quad & \frac{N \rightarrow N'}{(vn) N \rightarrow (vn) N'} \\
(\text{RSTRUCT}) \quad & \frac{N \equiv N' \quad N' \rightarrow M' \quad M' \equiv M}{N \rightarrow M} \quad (\text{RPAR}) \quad \frac{N \rightarrow M}{N \mid N' \rightarrow M \mid N'}
\end{aligned}$$

Table 1. Reduction Rules.

Example 2. Suppose a printer a has two functions: black-and-white and colour printing. The latter service is more expensive and therefore “harder” to get access to. The system is trust-based, meaning that according to its behaviour a principal may not be allowed to use a printer. In ctm this corresponds to writing principal $a\{P\}_\alpha$ where the policy and the protocol are defined as follows. Let message j represent the reception of a ‘junk document’ and \mathcal{M} be the interaction datatype of lists, where the predicate $a \cdot - \triangleright j < k$ checks that messages in the list of type $a \cdot \tilde{l} \triangleright j$ occur less than k times. We then define the policy π as $\{ \text{Access}(x, \text{Colour}) \leftarrow x \cdot - \triangleright j < 3; \text{Access}(x, \text{BW}) \leftarrow x \cdot - \triangleright j < 6 \}$ where $\text{Access}(x, y)$ is a predicate meaning that x can access y . Moreover we assume that $\text{upd}()$ keeps only lists of length at most n deleting the oldest messages and judging if a message is junk. Finally protocol P is defined as

$$P = !x \cdot \text{printC}(y) . \text{Access}(x, \text{Colour}) :: \text{printer} \cdot \text{printC}(y) \mid \\ !x \cdot \text{printBW}(y) . \text{Access}(x, \text{BW}) :: \text{printer} \cdot \text{printBW}(y)$$

In this example the action of granting access to the printer is modelled by sending a message to *printer*. A user could then be modelled as principal b running the protocol

$$Q = a \cdot \text{printC}(\text{spam}) . a \cdot \text{printBW}(\text{spam}) . a \cdot \text{printC}(\text{spam}) \mid a \cdot \text{printC}(\text{doc})$$

Suppose that $\text{upd}()$ will store *spam* as j and consider the network $N = a\{P\}_{(\pi, \emptyset)} \mid b\{Q\}_\alpha$ where \emptyset is the empty list. If $a \cdot \text{printC}(\text{doc})$ is executed first, b will get the authorisation to use the printer. But if the left component is all executed then b will no longer be able to colour-print as he has printed too much junk. Note that as we chose the function $\text{upd}()$ to keep lists of length at most n , any principal can behave well for n times and regain trust.

3 Barbed Equivalences

We now move to study the semantic theory of ctm . We first discuss the notion of observation formalised in terms of the actions offered to the environment. Formally we write $N \downarrow a \cdot b$ whenever one of the following conditions is satisfied:

- $N \downarrow a \cdot b$ if $N \equiv (v\tilde{n}) a\{ \phi :: b \cdot \tilde{l}(\tilde{m}) . P + P' \mid Q \}_\alpha \mid N'$ and $\alpha \vdash \phi, b \notin \mathbf{P}(N')$;
- $N \downarrow a \cdot b$ if $N \equiv (v\tilde{n}) a\{ p \cdot \tilde{l}(\tilde{x}) . P + P' \mid Q \}_\alpha \mid N'$ and $b \notin \mathbf{P}(N')$.

where $\mathbf{P}(N')$ is the set of principals contained in N' , $\tilde{l} \cap \tilde{n} = \emptyset$ and if $p \notin \mathbf{Var}$ then $p = a$. This definition excludes observing internal and restricted actions. Moreover we write $N \Downarrow a \cdot b$ whenever there exists M such that $N \rightarrow^* M$ and $M \downarrow a \cdot b$.

In the following we assume to work with closed protocols and networks.

Definition 3. A network barbed bisimulation is a symmetric relation \mathcal{R} on networks such that whenever $N \mathcal{R} M$

- $N \downarrow a \cdot b$ implies $M \downarrow a \cdot b$;
- $N \rightarrow N'$ implies $M \rightarrow M'$ and $N' \mathcal{R} M'$.

Two networks are barbed bisimilar ($\overset{\bullet}{\approx}$) if related by a network barbed bisimulation. Moreover we define $\overset{\bullet}{\approx}$ as above where \downarrow and \rightarrow after the two “implies” are substituted resp. by \Downarrow and \rightarrow^* .

3.1 Barbed Equivalences for Principals

We now define three different barbed equivalences for principals: one on protocols, one on policies and one on principals.

Protocol Congruence. Protocol barbed congruence compares only protocols. Contexts are, as usual, terms with a hole. We write $C_a[P]$ for the insertion of protocol P in the hole of context C , when the hole is placed in principal a .

Definition 4 (Protocol Barbed Congruence). Given a principal a , we say that P and Q are a -barbed congruent, written $P \simeq_a Q$, if $C_a[P] \dot{\simeq} C_a[Q]$ for all contexts $C_a[-]$.

Intuitively two protocols are congruent whenever they are able to observe the same events, input the same data and granting access in the same way, i.e. guards are such that there is no policy able to distinguish them. For instance, $\phi :: b \cdot l \langle m \rangle$ and $\phi' :: b \cdot l \langle m \rangle$ are equated only if ϕ and ϕ' hold true for exactly the same set of policies α .

Policy Equivalence. Varying the kind of contexts we use, we can use bisimulation to assess policies with respect to a fixed protocol P . The idea is that, given P , two policies are going to be equivalent whenever they “control” P 's behaviour in the same way.

Definition 5 (Policy equivalence). Given a principal a , we say that π and π' are a -barbed equivalent wrt P , written $\pi \simeq_a^P \pi'$, if for all contexts $C_a^P[\cdot] = a\{P\}_{(-,s)} \mid N$, we have $C_a^P[\pi] \dot{\simeq} C_a^P[\pi']$.

This notion allows, e.g., to remove formulas which P would never use.

Definition 6. We write

- $P \Downarrow \phi$ if $P \equiv (v\tilde{n}) (\phi :: b \cdot \tilde{l} \langle \tilde{m} \rangle . P + P' \mid P'')$ for $\tilde{n} \cap \tilde{l} = \emptyset$;
- $P \Downarrow \phi$ if there exists N and α such that $a\{P\}_\alpha \mid N \rightarrow^* a\{P'\}_{\alpha'} \mid N'$ and $P' \Downarrow \phi$;
- $P \Downarrow H$ if $H = \{\phi \mid P \Downarrow \phi\}$.

We can now state the following

Theorem 1. *Suppose that $P \Downarrow H$ and for all $\phi \in H$ and $s \in \mathcal{S}$ we have that $(\pi, s) \vdash \phi$ if and only if $(\pi', s) \vdash \phi$. Then $\pi \simeq_a^P \pi'$.*

The opposite is of course not true. Consider the protocol $P = \phi :: b \cdot l \langle m \rangle \mid \phi' :: b \cdot l \langle m \rangle$ and policies π and π' such that $(\pi, s) \vdash \phi$, $(\pi, s) \not\vdash \phi'$, $(\pi', s) \vdash \phi'$ and $(\pi', s) \not\vdash \phi$ for all s . In this case we have that $\pi \simeq_a^P \pi'$ but the policies entail different formulas wrt s .

Corollary 1. *Suppose that for all ϕ and s we have $(\pi, s) \vdash \phi$ if and only if $(\pi', s) \vdash \phi$. Then, $\pi \simeq_a^P \pi'$, for all P .*

In the following we write $\pi \vdash H$ whenever $H = \{\phi \mid (\pi, s) \vdash \phi \text{ for some } s\}$.

Theorem 2. *Suppose that $\pi \vdash H$ and $\pi' \vdash H'$. If $\pi \simeq_a^P \pi'$ for all P , then H and H' are equivalent, i.e. equal up to logical equivalence of the formulas they contain.*

Principal Equivalence. We now introduce the last of our equivalences which is the most general one.

Definition 7 (Principal Barbed Equivalence). Given a principal a , we say that (π, P) and (π', Q) are a -barbed equivalent, written $(\pi, P) \simeq_a^s (\pi', Q)$, if $C_a^s[\pi, P] \dot{\simeq} C_a^s[\pi', Q]$ for any context $C_a^s[-1, -2] = a\{ -2 \}_{(-1, s)} \mid N$.

It is possible to define the previous two equivalences in terms of barbed principal equivalence. We are now able to state the following

Proposition 1. $P \simeq_a Q$ if and only if for all s , π and protocols R we have that $(\pi, P \mid R) \simeq_a^s (\pi, Q \mid R)$.

Proposition 2. $\pi \simeq_a^P \pi'$ if and only if for all s we have that $(\pi, P) \simeq_a^s (\pi', P)$.

Example 3 (Implication). We consider a variation of the printer access control example and apply principal barbed equivalence. Suppose that a server a manages two printers both offering colour and b/w printing as before. The only difference between them is that Printer 1 does not distinguish colour from b/w printing, while Printer 2 does, e.g., by granting access only for b/w printing. For $z \in \{1, 2\}$ we define the following protocol for a print server.

$$\begin{aligned} P(z) = & (\nu n) (a \cdot n() \mid !a \cdot n\langle \rangle \cdot x \cdot z(y) \cdot \text{Access}(z, x) :: x \cdot z\langle \text{OK} \rangle \cdot \\ & (x \cdot z \cdot \text{col}() \cdot \text{Col}(z, x) :: x \cdot z \cdot \text{col}\langle \text{OK} \rangle \cdot a \cdot n() + \\ & x \cdot z \cdot \text{bw}() \cdot \text{BW}(z, x) :: x \cdot z \cdot \text{bw}\langle \text{OK} \rangle \cdot a \cdot n())) \end{aligned}$$

Note that the protocol first checks if it can give access to any type of printing, then verifies which one. The bang is used for writing a recursive protocol: after finishing dealing with a principal, the protocol will be ready once again to provide the service for printer z . The final server protocol is $P(1) \mid P(2)$; its policy is as below, where j and doc represents respectively a junk and a proper document.

$$\begin{aligned} \pi = & \{ \text{Access}(1, x) \leftarrow x \cdot 1 \triangleright j \leq x \cdot 1 \triangleright doc; \text{Col}(1, x) \leftarrow \text{Access}(1, x); \\ & \text{Col}(2, x) \leftarrow x \cdot 2 \triangleright j = 0; \text{BW}(1, x) \leftarrow \text{Access}(1, x); \text{BW}(2, x) \leftarrow \text{Access}(2, x); \\ & \text{Access}(2, x) \leftarrow x \cdot 2 \triangleright j \leq 5 \} \end{aligned}$$

Then the principal would be represented by the pair $(\pi, P(1) \mid P(2))$. Using the equivalence \simeq_a^s we can rewrite the principal as $(\pi, Q \mid P(2))$ where

$$\begin{aligned} Q = & (\nu n) (a \cdot n() \mid !a \cdot n\langle \rangle \cdot x \cdot 1(y) \cdot \text{Access}(1, x) :: x \cdot 1\langle \text{OK} \rangle \cdot \\ & (x \cdot 1 \cdot \text{col}() \cdot x \cdot 1 \cdot \text{col}\langle \text{OK} \rangle \cdot a \cdot n() + x \cdot 1 \cdot \text{bw}() \cdot x \cdot 1 \cdot \text{bw}\langle \text{OK} \rangle \cdot a \cdot n())) \end{aligned}$$

In fact $(\pi, P(1) \mid P(2)) \simeq_a^s (\pi, Q \mid P(2))$ for any s and this can be explained by the following argument. In protocol $P(1) \mid P(2)$ the output $x \cdot 1 \cdot \text{Col}\langle \text{OK} \rangle$ is guarded by $\text{Col}(1, x)$ instead in $Q \mid P(2)$ it is unguarded. We need to show that $\text{Col}(1, x)$ is always true at that point of the computation. In fact in π the predicate $\text{Access}(1, x)$ implies $\text{Col}(1, x)$ and the action $x \cdot 1 \cdot \text{bw}$ does not change the value of $\text{Access}(1, x)$ (Q and $P(2)$ are sequential) as well as the inputs executed by the branch $P(2)$.

We can also use our previous results: by Theorem 1 we have that $\pi \simeq_a^{OIP(2)} \pi'$ where

$$\pi' = \{ \text{Access}(1, x) \leftarrow x \cdot 1 \triangleright j \leq x \cdot 1 \triangleright \text{doc}; \text{Col}(2, x) \leftarrow x \cdot 2 \triangleright j = 0; \\ \text{Access}(2, x) \leftarrow x \cdot 2 \triangleright j \leq 5 \}.$$

Now by Proposition 2 we then have that $(\pi, P(1) \mid P(2)) \simeq_a^s (\pi', Q \mid P(2))$ for all s .

3.2 A Barbed Congruence for Networks

Above we have analysed equivalences for manipulating single principals. But clearly we also need techniques to reason about networks forming webs of trust. For this purpose, we now introduce a barbed congruence for networks.

Definition 8 (Network Barbed Congruence). We say that two networks N and M are barbed congruent, written $N \approx M$, if for any context $C[-]$ $C[N] \dot{\approx} C[M]$.

The above definition defines a congruence which is well known as weak barbed congruence. We now show an interesting application of such a congruence in an example where recommendations are taken into account.

Example 4 (Recommendations and Trust Security). Trusting someone may be a consequence of observed good behaviour, but it may also be a consequence of good recommendations from a trusted third party. In this example we show how to describe in `ctm` a system where principals' trust is based both on observations and recommendations. We consider a European network of banks where each bank issues mortgages for customers according to a policy. The policy grants a mortgage whenever the customer has always paid back previous mortgages and, additionally, other banks' opinions about the customer are positive. In `ctm`, granting the mortgage is equivalent to proving a predicate $G(x)$ from the following policy

$$\pi_1(Y) = \{G(x) \leftarrow Y \cdot - \triangleright (x, \text{Bad}) = 0, \mathbb{M}(x); \text{Good}(x) \leftarrow \mathbb{M}(x), \text{Bad}(x) \leftarrow \text{NoM}(x)\}.$$

The interaction datatype is multiset. The predicates `Good` and `Bad` (which will be used for recommendations) depend on the predicate `M` and `NoM` only (local observations), instead the predicate `G` depends also on the recommendations received from Y . The predicate $\mathbb{M}(x)$ is assumed to check whether every mortgage granted to x is matched by a full repayment: in order to get a mortgage, there must be no outstanding mortgages. This is expressed by identifying messages with a fresh name w . We can then define the following template for banks.

$$P_1(X, Y) = !x \cdot \text{mg}(w) \cdot (\nu k) (Y \cdot \text{rec}(k, x) \cdot Y \cdot k(x, z) \cdot G(x) :: X \cdot \text{gr}(x, w)) \\ | !X \cdot \text{gr}(x, w) \cdot x \cdot w() \cdot x \cdot w() \\ | !Y \cdot \text{rec}(k, x) \cdot (G(x) :: Y \cdot k(x, \text{Good}) + \text{Bad}(x) :: Y \cdot k(x, \text{Bad}))$$

Bank X has three components: one for granting mortgages, one for accounting, and one for giving recommendations to another bank Y . When receiving a request from a on channel `mg` a fresh name w reserved for the particular transaction is received. Then

X will send a request of recommendation to bank Y . At this point, if the predicate G is provable from the policy, the protocol will transfer the request to the accounting component (on channel gr), which will send an authorisation message to a , and will finally be waiting for a repayment. In case G is not provable, the request will be pending. As pointed out before, the policy will take care of denying further authorisations until repayment. The third component, which gives recommendations to bank Y , just checks whether the predicate Good or Bad are provable, and sends a message accordingly. Suppose now that B_F and B_I are respectively French and Italian banks. We can define a network of banks as follows.

$$N_1 = B_F\{ P_1(B_F, B_I) \}_{(\pi_1(B_I), \emptyset)} \mid B_I\{ P_1(B_I, B_F) \}_{(\pi_1(B_F), \emptyset)}$$

Suppose now that a customer a has just moved from Italy to France, and she is asking the French bank B_F for a mortgage, e.g. with the protocol $(\nu w) B_F \cdot \text{mg}\langle w \rangle \cdot B_F \cdot w() \cdot B_F \cdot w\langle \rangle$. Let us now define a different network of banks using a third party (O) which is going to deal with all the requests. The following policy is used by principal O .

$$\pi_2(X, Y) = \{G(X, x) \leftarrow M(Y, x), M(X, x)\}$$

All the operations previously performed by the banks will now be performed by this policy, and we no longer need recommendations. The following is part of the protocol for principal O .

$$F(W) = W(x, w) \cdot G(W, x) :: O \cdot W \cdot \text{gr}\langle x, w \rangle \mid !O \cdot W \cdot \text{gr}\langle x, w \rangle \cdot W \cdot w\langle \rangle \cdot W \cdot w()$$

The banks will only forward messages from the principals to O and vice-versa.

$$P_2 = !x \cdot \text{mg}\langle w \rangle \cdot O\langle x, w \rangle \cdot O \cdot w() \cdot x \cdot w\langle \rangle \cdot x \cdot w() \cdot O \cdot w\langle \rangle$$

The entire new network will be

$$N_2 = O\{ !(F(B_F) + F(B_I)) \}_{(\pi_2(B_F, B_I) \cup \pi_2(B_I, B_F), \emptyset)} \mid B_F\{ P_2 \}_{(\emptyset, \emptyset)} \mid B_I\{ P_2 \}_{(\emptyset, \emptyset)}$$

We then have that $N_1 \approx N_2$ accordingly to the definition of \approx .

In this example, O works as a “headquarter” which collects information from the banks. This is a further step from our previous work [8] where we computed principal trust policies as the least fixed point of a global function. Such a thing, plainly unfeasible in a distributed scenario, can be implemented in ctm . For instance, the global trust function can be expressed as the policy of a principal, e.g. O , and the fixed point as a computation. Then, using the network equivalence, one may be able to simplify the network and avoid the use of “headquarters,” like in this example. This is generic technique for stating (and proving) the correctness of a “web of trust”, with a specification in the form of a centralised “headquarter”.

4 On the expressive power of global input

Our calculus uses a new input construct: global input. In this section we prove that such a construct adds expressiveness to the language. Let $\text{ctm}^{-\phi}$ be the fragment of ctm

where all inputs are guarded by \mathbf{tt} and let $\mathbf{ctm}^{-x;\phi}$ be the fragment of $\mathbf{ctm}^{-\phi}$ without global input. Moreover let S be a list of observations $a_1 \cdot b_1; \dots a_k \cdot b_k; \dots$ and $N \Downarrow S$ if and only if $N \rightarrow^* N_1 \rightarrow^* \dots N_k \rightarrow^*$ and $N_1 \Downarrow a_1 \cdot b_1, \dots, N_k \Downarrow a_k \cdot b_k, \dots$. In the following, with abuse of notation, we will use $\llbracket - \rrbracket$ for both networks and protocols.

Definition 9. An encoding $\llbracket - \rrbracket : \mathbf{ctm}^{-\phi} \longrightarrow \mathbf{ctm}^{-x;\phi}$ is sensible whenever for all protocols P, Q and networks N, M

- $\llbracket P \mid Q \rrbracket = \llbracket P \rrbracket \mid \llbracket Q \rrbracket$
- $\llbracket N \mid M \rrbracket = \llbracket N \rrbracket \mid \llbracket M \rrbracket$
- $\llbracket a\{ P \}_\alpha \rrbracket = a\{ \llbracket P \rrbracket \}_\alpha$
- for any N , $a\{ P \}_\alpha \mid N \Downarrow S$ if and only if $\llbracket a\{ P \}_\alpha \mid N \rrbracket \Downarrow S$

The first three rules represent the notion of uniform encoding, while the last one corresponds to the notion of reasonable encoding according to [15].

Theorem 3. There is no sensible encoding $\llbracket - \rrbracket$ from $\mathbf{ctm}_p^{-\phi}$ into $\mathbf{ctm}_p^{-x;\phi}$.

Proof. [Sketch] Suppose there exists such an encoding and consider $a\{ x \cdot l(y) \}_\alpha$. Principal a is such that $a\{ x \cdot l(y) \}_\alpha \Downarrow a \cdot b$ for any b . Now we have that $a\{ \llbracket x \cdot l(y) \rrbracket \}_\alpha \not\Downarrow a \cdot b$ for all b . In fact we can prove by induction on the protocols of $\mathbf{ctm}_p^{-x;\phi}$ that such a protocol does not exist. \square

5 Conclusion

We have introduced \mathbf{ctm} , a calculus for trust management. The calculus enjoys many new features which fit in global computing scenarios making use of the notion of trust.

Principals in \mathbf{ctm} have two components: the policy and the protocol. The policy consists of an immutable part, α , and a variable s . The former expresses the logic of the policy, i.e. the rules following which decisions are taken, on the basis of past experiences. The latter records the observations which make up such experiences, as a function of the messages exchanged in interactions between principals.

It may be objected that this yields a generic concurrent calculus of stateful entities, and not a calculus specifically designed to represent trust-based systems. This is actually not the case. The key to the matter is that, while s is definitely a kind of store, principals have absolutely no control as to what it stores, or when it stores it: s is updated uniquely and exactly to reflect the outcome of interactions. These include feedback on untrusted clients and advice from trusted principals. In particular, a principal cannot store arbitrary values to s , or retrieve them from it. In other words, the calculus represents faithfully a distributed set of principals interacting with each other according to trust policies and risk assessment based on computational histories. Similarly it is not possible to compare \mathbf{ctm} to an extension to locations of the applied π -calculus [1] as the latter does not model the notion of collecting observations even though function guards can represent policies.

We remark also that our use of guards works quite effectively with the choice of synchronous communications, to abstract the sequence of actions service request, risk

assessment, response to client, and record observation, in a single, atomic step where trust-based decisions are localised.

The equivalences for `ctm` are interesting but still lack efficient proof methods. In order to accomplish this we aim at defining a labelled transition system for characterising all the equivalences studied. It would also be interesting to treat static analysis for the calculus, e.g. a type system, and study its relationship with what shown in this paper.

Acknowledgements. We thank J. Almansa, S. Agarwal, B. Klin, K. Krukow and P. Oliva for useful comments. Thanks also goes to the anonymous referees for their helpful suggestions.

References

1. M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proc. of the 28th symposium on Principles of Programming Languages (POPL'01)*, pages 104–115. ACM Press, 2001.
2. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of databases*. Addison-Wesley, 1995.
3. R. Amadio, G. Boudol, and C. Lhoussaine. The receptive distributed pi-calculus. In *Proc. of the 19th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS '99)*, volume 1738 of LNCS, pages 304–315. Springer-Verlag, 1999.
4. A. W. Appel and E. W. Felten. Proof-carrying authentication. In *Proc. of 6th ACM Conference on Computer and Communications Security (CCS '99)*, 1999.
5. M. Y. Becker and P. Sewell. Cassandra: Flexible trust management, applied to electronic health records. In *Proc. of the 17th IEEE Computer Security Foundations Workshop (CSFW '04)*. IEEE Computer Society Press, 2004.
6. M. Burrows, M. Abadi, B. W. Lampson, and G. Plotkin. A calculus for access control in distributed systems. In *Proc. of 11th Annual International Cryptology Conference Advances in Cryptology (CRYPTO '91)*, volume 576, pages 1–23, 1991.
7. M. Carbone and S. Maffies. On the expressive power of polyadic synchronisation in π -calculus. *Nordic Journal of Computing (NJC)*, 10(2), September 2003.
8. M. Carbone, M. Nielsen, and V. Sassone. A formal model for trust in dynamic networks. In *Proc. of International Conference on Software Engineering and Formal Methods (SEFM'03)*, pages 54–61. IEEE Computer Society Press, 2003.
9. J. Guttman, J. Thayer, J. Carlson, J. Herzog, J. Ramsdell, and B. Sniffen. Trust management in strand spaces: A rely-guarantee method. In *Proc. of the European Symposium on Programming (ESOP '04)*, LNCS, pages 325–339. Springer-Verlag, 2004.
10. M. Hennessy and J. Riely. Resource access control in systems of mobile agents. *Information and Computation*, 173(1):82–120, 2002.
11. S. Jajodia, P. Samarati, and V. S. Subrahmanian. A logical language for expressing authorizations. In *Proc. of the IEEE Symposium on Security and Privacy*, pages 31–42. IEEE Computer Society Press, 1997.
12. A. J. I. Jones and B. S. Firozabadi. On the characterisation of a trusting agent. In *Workshop on Deception, Trust and Fraud in Agent Societies*, 2000.
13. N. Li, J. C. Mitchell, and W. H. Winsborough. Design of a role-based trust management framework. In *Proc. of the IEEE Symposium on Security and Privacy*, pages 114–130. IEEE Computer Society Press, 2002.
14. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I and II. *Information and Computation*, 100(1):1–40,41–77, September 1992.
15. C. Palamidessi. Comparing the expressive power of the synchronous and the asynchronous π -calculus. In *Proc. of the 24th symposium on Principles of Programming Languages (POPL'97)*, pages 256–265. ACM Press, 1997.