Distributed Approximation of Fixed-Points in Trust Structures[*]

Karl Krukow[†]
BRICS[‡]
University of Aarhus
Aarhus, Denmark
krukow@brics.dk

Andrew Twigg[§]
Computer Laboratory
University of Cambridge
Cambridge, UK
Andrew.Twigg@cl.cam.ac.uk

Abstract

We consider distributed algorithms for solving a range of problems in a framework for trust in large-scale distributed systems. The framework is based on the notion of trust structures; a set of 'trust-levels' with two distinct partial orderings. In the trust model, a global trust-state is defined as the least fixed-point of a collection of local policies of nodes in the network.

We show that it is possible to compute the global trust-state using a simple, robust and totally asynchronous distributed-algorithm. We also consider a distributed notion of proof-carrying-requests as a means of approximating the least fixed-point, enabling sound reasoning about the global trust-state without computing the exact fixed-point. Our proof-carrying-request model is different than the notion of proof-of-compliance from traditional trust-management; in particular, all proofs are efficiently verifiable or easily rejected, but may require as much communication as computing the actual trust-state itself, in the worst-case.

1 Introduction

This paper completes a mathematical model for trust in large-scale distributed systems, recently introduced by Carbone, Nielsen and Sassone [8].

The need for flexible security mechanisms in emerging distributed-systems is evident. However, the diversity and scale of such systems, combined with the lack of centralized authority, means that traditional mechanisms for security decision-making, e.g. access-control lists, are often too restrictive and complex to deploy [2]. The concept of trust management, introduced by Blaze *et al.* [4], was presented as a solution to the problems with authorization in large-scale distributed systems. Traditional trust-management systems make security decisions based on policies, dealing with authorization by deciding the so-called compliance-checking problem: given a *request* to perform a certain action, together with a set of *credentials*; does the request comply with the local *security policy*, given the credentials?

In *dynamic* trust-management-systems [16, 22, 15], trust-specifications are often based on the *past behaviour* of principals, which gives rise to a different, more flexible notion of trust than that of traditional trust-management systems. The traditional systems often take an "all or nothing" approach, in which no or partial credentials necessarily means no interaction. By broadening the range of specifications of trust-levels, one may encourage interaction in situations where the traditional approach would be too restrictive.

While the traditional notion of trust management is well understood, e.g. Mitchell *et al.* [9, 18], and, to a large extent, captured concisely in a mathematical framework of Weeks [24]; a lot of the "broader" dynamic systems lack such foundation in formal methods (this point is illustrated by the wide range of related systems in the survey [13]). This lack prompted the development of a mathematical framework for trust [8], inspired by that of Weeks, but departing from Weeks by emphasizing the concept of *information* in contrast to *authorization*. The framework, which was introduced by Carbone *et al.* [8], discussed also by Nielsen *et al.* [19] and Krukow [15], is the focus of this paper. To understand our motivation and contributions, one must first understand this trust model, and, consequently, we describe the model now.

[*]Extended abstract of BRICS technical report, RS-05-6, http://www.brics.dk/RS/05/6.

[†]Supported by SECURE: Secure Environments for Collaboration among Ubiquitous Roaming Entities, EU FET-GC IST-2001-32486. Supported by DISCO: Semantic Foundations of Distributed Computing, EU IHP, Marie Curie, HPMT-CT-2001-00290.

[‡]BRICS: Basic Research in Computer Science (www.brics.dk), funded by the Danish National Research Foundation.

[§]Supported by British Telecommunications Plc and a Marie Curie Fellowship HPMT-CT-2000-00093.

1.1 Towards a Formal Model for Trust

A *trust model* is a mathematical model which gives precise meaning to the (otherwise overloaded) concept of *trust* within a system or a class of systems. A trust model should be generic enough to be instantiated to support authorization in a variety of distributed computing systems. For example, in a P2P file-sharing system, appropriate authorizations may include using resources 'download' and 'upload', while in the PGP system [21], a principal may be authorized to introduce new signed $(\mathit{key}, \mathit{name})$ pairs.

The trust-structure framework [8] is a generic model, parameterized by a set X of possible *trust values* representing distinct levels or degrees of trust, relevant for a particular application. For example, in the P2P file-sharing application, one might identify a trust-level with an authorization, say $X_{P2P} = \{\mathsf{upload}, \mathsf{download}, \mathsf{no}, \mathsf{both}, \mathsf{unknown}\}$. Here a principal p might assign trust-value 'upload' to q, while, since it doesn't know r, p assigns value 'unknown' to r; at the same time, c is known never to be trusted and hence is assigned value 'no'. In the trust-structure framework, trust levels are not always identified with *authorizations*, e.g., in the P2P scenario one could instead use more "dynamic" trust values, related to the *past behaviour* of principals; for instance $X'_{P2P} = \{(ul, dl) \mid ul, dl \in \mathbb{N}\}$, where $(m, n) \in X'_{P2P}$ represents the past history of a principal that has performed m uploads and n downloads.

The trust-structure framework simply assumes that X is a set for which it makes sense to (partially) order its values in two ways: with respect to *more trust* and with respect to *more information*. For example, in X_{P2P} the value 'no' clearly denotes a lower degree of trust than 'download', which is reflected by the *trust ordering* $\preceq$, e.g. we have $\mathsf{no} \preceq \mathsf{download}$. The ordering $\preceq$ is *partial*, meaning that it may not make sense to relate all pairs of trust values, e.g. relating $\mathsf{download}$ and upload is not meaningful, so neither $\mathsf{download} \preceq \mathsf{upload}$ nor $\mathsf{upload} \preceq \mathsf{download}$. An important characteristic of the trust structure framework is the requirement that not only does it make sense to order values according to the trust ordering ($\preceq$), but values are also partially ordered with respect to *information*. Since we are allowing various degrees of precision (or information) in the trust values, it makes sense to compare some values with respect to their information content, e.g. $\mathsf{unknown}$ is clearly less information than upload or no. In general, the framework assumes that the set X can be partially ordered by $\sqsubseteq$, called the information ordering. One may think of assertion $x \sqsubseteq y$ as the statement that x can be refined into y, or that x approximates y, e.g. 'unknown' could be refined into 'no' if more (trust-wise negative) information was provided.

We provide now the formal definitions of the trust-structure framework. A complete formal understanding of the framework requires an understanding of the theory of partial orders as can be obtained, e.g., in Winskel's book on programming language semantics [25]. The casual reader should be able to understand the model at a more intuitive level from the following description.

Trust structures. Formally, trust is something which exists between *pairs of principals*; it is *quantified* and *asymmetric* in that we care of "how *much*" or "to what *degree*" principal p trusts principal q (which may not be to the same degree that q trusts p). Each application instance of the framework defines a so-called *trust structure*, $T = (X, \preceq, \sqsubseteq)$, which consists of a set X of *trust values*, together with two partial orderings of X, the trust ordering ($\preceq$) and the information ordering ($\sqsubseteq$). The elements $s, t \in X$ express the levels of trust that are relevant for the particular instance, and $s \preceq t$ means that t denotes at least as high a trust-level as s. As we have seen, in contrast, the information ordering introduces a notion of precision or refinement. As a simple example of a trust structure, consider the so-called "MN" trust-structure T_{MN} [15]. In this structure, trust values are pairs (m, n) of natural numbers (as in the set X'_{P2P}), representing $m + n$ interactions with a principal; each interaction classified as either "good" or "bad". In a trust value (m, n), the first component, m, denotes the number of "good" interactions, and the second, the number of "bad" ones. The information-ordering is given by: $(m, n) \sqsubseteq (m', n')$ only if one can refine (m, n) into (m', n') by adding zero or more good interactions, and, zero or more bad interactions, i.e., iff $m \le m'$ and $n \le n'$. In contrast, the trust ordering is given by: $(m, n) \preceq (m', n')$ only if $m \le m'$ and $n \ge n'$. Nielsen *et al.* [15, 20, 8], have considered several additional examples of trust structures.

Global trust-states. Given a fixed trust structure $T = (X, \preceq, \sqsubseteq)$, and a set $\mathcal{P}$ of principal identities; a *global trust-state* of the system is a function $\mathsf{gts} : \mathcal{P} \to \mathcal{P} \to X$. The interpretation is that gts represents the trust state where p's trust in q (formalized as an element of X) is given by $\mathsf{gts}(p)(q)$. A good way of thinking about gts is to consider it a large matrix, indexed by pairs of principal identities, in which the row indexed by principal p (denoted $\mathsf{gts}(p)$) contains principal p's trust in any other principal. For example, in the row $\mathsf{gts}(p)$, column q represents p's trust in q, given as an element in the set X; this entry is denoted $\mathsf{gts}(p)(q)$ ("row vectors" like $\mathsf{gts}(p)$ are also called *local* trust-states). Thus, the matrix gts gives a complete (system global) description of how everyone trusts everyone else. We shall write $\mathtt{GTS}$ for the set of global trust-states $\mathcal{P} \to \mathcal{P} \to X$. Similarly we write $\mathtt{LTS}$ for the set $\mathcal{P} \to X$ of *local* trust-states (corresponding to rows of gts matrices).

Trust policies. The goal of the trust-structure framework is to define, at any time, a global trust state $\overline{\mathsf{gts}}$, thus giving a precise meaning to "p's trust in q" at all times (i.e., as value $\overline{\mathsf{gts}}(p)(q)$). In order to uniquely define the global trust state $\overline{\mathsf{gts}}$, an approach similar to that of Weeks [24] is adopted. Each principal $p \in \mathcal{P}$ defines a *trust policy* which is a function π_p of type $\mathtt{GTS} \rightarrow \mathtt{LTS}$, i.e. taking a global matrix as input, and providing a local row-*vector* as output. This function then determines p's trust-row within the unique global trust-matrix, i.e. determines row $\overline{\mathsf{gts}}(p)$, as follows. In the simplest case, π_p could be a constant function, ignoring its first argument $\mathsf{gts} : \mathcal{P} \rightarrow \mathcal{P} \rightarrow X$. As an example, $\pi_p(\mathsf{gts}) = \lambda q.t_0$ (for some $t_0 \in X$) defines p's trust in any $q \in \mathcal{P}$ as the constant t_0. In general we allow a form of *delegation* called *policy reference*: policy π_p may refer to other policies (π_z, $z \in \mathcal{P}$), e.g., p might trust q to download if A or B trusts q to download. The general interpretation of π_p is the following. *Given that all principals assign trust-values as specified in the global trust-state* gts, *then p assigns trust values as specified in vector* $\pi_p(\mathsf{gts}) : \mathcal{P} \rightarrow X$. For example, in the X_{P2P} trust structure, function $\pi_p(\mathsf{gts}) = \lambda q \in \mathcal{P}.(\mathsf{gts}(A)(q) \vee_{\preceq} \mathsf{gts}(B)(q)) \wedge_{\preceq} \mathsf{download}$, represents a policy saying "for any $q \in \mathcal{P}$, the trust in q is the least upper-bound in $(X_{P2P}, \preceq)$ of what A and B say, but no more than the constant $\mathsf{download} \in X_{P2P}$."[1]

Unique trust-state. The collection of all trust policies, $\Pi = (\pi_p | p \in \mathcal{P})$, thus "spins a global web-of-trust" in which the trust policies mutually refer to each other. Since trust policies Π may give rise to cyclic policy-references, it is not *a priori* clear how to define the unique global trust-state $\overline{\mathsf{gts}}$ for a given collection of trust policies Π. One may consider the unique function $\Pi_\lambda = \langle \pi_p | p \in \mathcal{P} \rangle$, of type $\mathtt{GTS} \rightarrow \mathtt{GTS}$ with the property that $\mathtt{Proj}_p \circ \Pi_\lambda = \pi_p$ for all $p \in \mathcal{P}$, where $\mathtt{Proj}_p$ is the p'th projection.[2] Intuitively, the function Π_λ is easy to understand: each π_p maps a matrix $\mathsf{gts} \in \mathtt{GTS}$ to a "row-vector" $\pi_p(\mathsf{gts})$ in $\mathtt{LTS}$; on input gts, function Π_λ builds the output matrix from all these rows by taking the p'th row of the output matrix to be $\pi_p(\mathsf{gts})$. We can now state a minimal requirement that the unique trust state, $\overline{\mathsf{gts}}$, should satisfy: $\overline{\mathsf{gts}}$ *should be consistent with all policies* π_p. This amounts to requiring that it should satisfy the following fixed-point equation: $\mathsf{gts}(p) = \pi_p(\mathsf{gts})$ for all $p \in \mathcal{P}$; or equivalently:

$$\Pi_\lambda(\mathsf{gts}) \quad = \quad \mathsf{gts}$$

Any matrix $\mathsf{gts} : \mathtt{GTS}$ satisfying this equation is *consistent* with the policies $(\pi_p | p \in \mathcal{P})$, i.e. row p of gts is consistent with π_p in that, if all principals trust as specified in gts, then p trusts as specified in $\pi_p(\mathsf{gts})$ which (by the fixed-point equation) can be read-off as the pth row of gts.

This means that *any* fixed point of Π_λ is consistent with all policies π_p. But arbitrary functions Π_λ, may have multiple or even no fixed points.

Here we appeal to the power of the mathematical theory of complete partial orders and continuous functions (domain theory), known from formal programming language semantics [25]. A crucial requirement in the trust-structure framework is that the information ordering $\sqsubseteq$ makes $(X, \sqsubseteq)$ a complete partial order (cpo) with a least element (this element is denoted $\perp_\sqsubseteq$, and can be thought of as a value representing "unknown"). We require also that all policies $\pi_p : \mathtt{GTS} \rightarrow \mathtt{LTS}$ are *information continuous*, i.e. continuous with respect to $\sqsubseteq$.[3] Since this implies that Π_λ is also information-continuous, and since $(\mathtt{GTS}, \sqsubseteq)$ is a cpo with bottom, standard theory [25] tells us that Π_λ has a (unique) least fixed-point which we denote $\mathsf{lfp}_\sqsubseteq \Pi_\lambda$ (or simply $\mathsf{lfp}\, \Pi_\lambda$):

$$\mathsf{lfp}_\sqsubseteq \Pi_\lambda = \bigsqcup_\sqsubseteq \{\Pi_\lambda^i(\lambda p.\lambda q.\perp_\sqsubseteq) \mid i \in \mathbb{N}\}$$

This global trust-state has the property that it is a fixed-point (i.e., $\Pi_\lambda(\mathsf{lfp}_\sqsubseteq \Pi_\lambda) = \mathsf{lfp}_\sqsubseteq \Pi_\lambda$) and that is is the (information-) least among fixed-points (i.e., for any other fixed point gts, $\mathsf{lfp}_\sqsubseteq \Pi_\lambda \sqsubseteq \mathsf{gts}$). Hence, for any collection Π of trust policies, we can define the *global trust-state induced by that collection*, as $\overline{\mathsf{gts}} = \mathsf{lfp}\, \Pi_\lambda$, which is well-defined by uniqueness.

Consider now two mutually referring functions π_p and π_q, given by $\pi_p(\mathsf{gts}) = \mathtt{Proj}_q(\mathsf{gts})$, and $\pi_q(\mathsf{gts}) = \mathtt{Proj}_p(\mathsf{gts})$. Intuitively, there is no information present in these functions; p delegates all trust-questions to q, and similarly q delegates to p. In this case, we would like the global trust-state $\overline{\mathsf{gts}}$ induced by the functions to take the value $\perp_\sqsubseteq$ on any entry $z \in \mathcal{P}$ for both p and q, i.e., for both $x = p$ and $x = q$ and for all $z \in \mathcal{P}$ we should have $\overline{\mathsf{gts}}(x)(z) = \perp_\sqsubseteq$. This is exactly what is obtained by choosing the information-*least* fixed-point of Π_λ.

1.2 Motivation and Technical Contributions

Many interesting systems are instances of the trust-structure framework [8, 15], but one could argue against its usefulness as a basis for the actual construction of trust-management systems. In order to make security decisions, each principal p will need to reason about its trust in others, that is, the values of $\overline{\mathsf{gts}}(p)$. While the framework does ensure the existence of a unique (theoretically well-founded)

[1]Assuming that $(X, \preceq)$ is a lattice. We always denote information ($\sqsubseteq$) least-upper-bounds by "square" symbols $\sqcup$, and trust ($\preceq$) least-upper-bounds/greatest-lower-bounds by $\vee/\wedge$.

[2]$\mathtt{Proj}_p$ is given by: for all $\mathsf{gts} : \mathcal{P} \rightarrow \mathcal{P} \rightarrow X$. $\mathtt{Proj}_p(\mathsf{gts}) = \mathsf{gts}(p)$.

[3]We overload $\sqsubseteq$ (respectively $\preceq$) to denote also the pointwise extension of $\sqsubseteq$ ($\preceq$) to the function space $\mathtt{LTS} = \mathcal{P} \rightarrow X$ as well as to $\mathtt{GTS} = \mathcal{P} \rightarrow \mathcal{P} \rightarrow X$. Saying that a policy is information-continuous means that the function is continuous w.r.t. $\sqsubseteq$.

global trust-state, it is not "operational" in the sense of providing a way for principals to actually *compute* the trust values. Furthermore, as we shall argue in the following, the standard way of computing least fixed-points is inadequate in our scenario.

When the cpo $(X, \sqsubseteq)$ is of finite height h, the cpo $(\mathcal{P} \to \mathcal{P} \to X, \sqsubseteq)$ has height $|\mathcal{P}|^2 \cdot h$.[4] In this case, the least fixed-point of Π_λ can, *in principle*, be computed by finding the first identity in the chain of approximants $(\lambda p.\lambda q.\bot_\sqsubseteq) \sqsubseteq \Pi_\lambda(\lambda p.\lambda q.\bot_\sqsubseteq) \sqsubseteq \Pi_\lambda^2(\lambda p.\lambda q.\bot_\sqsubseteq) \sqsubseteq \cdots \sqsubseteq \Pi_\lambda^{|\mathcal{P}|^2 \cdot h}(\lambda p.\lambda q.\bot_\sqsubseteq)$ [25]. However, in the environment envisioned, such a computation is infeasible. The functions $(\pi_p : p \in \mathcal{P})$ defining Π_λ are distributed throughout the network, and, more importantly, even if the height h is finite, the number of principals $|\mathcal{P}|$, though finite, will be *very* large. Furthermore, even if resources were available to make this computation, we can not assume that any central authority is present to perform it. Finally, since each principal p defines its trust policy π_p autonomously, an inherent problem with trying to compute the fixed point is the fact that p might decide to change its policy π_p to π'_p at any time. Such a policy update would be likely to invalidate data obtained from a fixed-point computation done with global function Π_λ, i.e., one might not have time to compute $\mathsf{lfp}\,\Pi_\lambda$ before the policies have changed to Π'.

The above discussion indicates that exact computation of the fixed point is infeasible, and hence that the framework is not suitable as an operational model. Our motivation is to counter this by showing that the situation is not as hopeless as suggested. The rest of the paper presents a collection of techniques for *approximating* the *idealized* fixed-point $\mathsf{lfp}\,\Pi_\lambda$. Our work essentially deals with the operational problems left as "future work" by Carbone *et al.* [8]. More specifically, this consists of three operational issues.

Firstly, techniques for actual distributed computation of approximations to the idealized trust-values, over a global, highly dynamic, decentralized network. We start by showing that although it may be infeasible to compute the global trust-state, $\overline{\mathsf{gts}} : \mathcal{P} \to \mathcal{P} \to X$, one can instead try to compute so-called *local* fixed-point values. We take the practical point-of-view of a specific principal R, wanting to reason about its trust value for a fixed principal q. The basic idea is that instead of computing the entire state $\overline{\mathsf{gts}}$, and *then* "looking up" value $\overline{\mathsf{gts}}(R)(q)$ to learn R's trust in q, one may instead compute this value directly. We prove a convergence result that enables us to apply a robust totally-asynchronous distributed algorithm of Bertsekas [1] for local fixed-point computation. This is developed in Section 2.

Secondly, often it is infeasible and even unnecessary to compute the *exact* denotation of a set of policies. In many cases it is sufficient (in order to make a trust-based decision) to know that a certain property of this value is satisfied. In Section 3, we take very mild assumptions on the relation between the two orderings in trust structures. This enables us to prove the soundness of two efficient protocols for safe approximation of the least fixed-point. Often this allows principals to take security-decisions without having to compute the exact fixed-point. For example, suppose we know a function $\bar{p} : \mathcal{P} \to \mathcal{P} \to X$ with the property that $\bar{p} \preceq \overline{\mathsf{gts}}$. In many trust structures it is the case that if $\bar{p}$ is sufficient to authorize a given request, so is the actual fixed-point.

Finally, the inherently dynamic nature of the envisioned systems requires algorithms that explicitly deal with the dynamic updating of trust policies (rather than implicitly dealing with updates by doing a *complete* re-computation of the trust-state). In the full paper [17], we address the problem of dynamic policy-changes. We provide algorithms that reuse information from "old" computations, when computing the "new" fixed-point values. For specific (but commonly occurring) types of updates this is very efficient. For fully general updates we have an algorithm which is better than the naive algorithm in many cases.

Future and related work is discussed in the concluding section. The full paper [17] contains proofs of all theorems, detailed descriptions of the algorithms, and examples illustrating the algorithms.

2 Computation of Least Fixed-Points

In this section, we show how to compute the *local* fixed-point value $\overline{\mathsf{gts}}(R)(q)$ for two fixed principals R and q, without computing the complete global trust-state $\overline{\mathsf{gts}}$. The reason for computing local values is twofold. First, we can benefit from distributing the computational- and storage-burdens, so that instead of centrally computing the complete state $\overline{\mathsf{gts}}$, node will R maintain "entry" $\overline{\mathsf{gts}}(R)(q)$ in the "distributed matrix" $\overline{\mathsf{gts}}$. Second, although the semantics of trust policies are functions of type $(\mathcal{P} \to \mathcal{P} \to X) \to \mathcal{P} \to X$ which (due to policy referencing) in general may depend on the trust values of *all* principals, we expect that in practice, policies will not be written in this way. Instead, policies are likely to refer to a few known (and usually "trusted") principals. For fixed R and q, the set of principals that R's policy *actually* depends on in its entry for q, is often a significantly smaller subset of $\mathcal{P}$. For example, our policy from the previous section, $\pi_R(\mathsf{gts}) = \lambda q \in \mathcal{P}.(\mathsf{gts}(A)(q) \vee_\preceq \mathsf{gts}(B)(q)) \wedge_\preceq \mathsf{download}$, is independent of all entries of gts except for those of principals A and B. This means that in order to evaluate π_R with respect to some principal q, R needs only information from A and B.

We first compute (distributedly) a dependency graph

[4] The height of a cpo is the size of its longest chain.

which contains *only the dependencies relevant for the computation of* $\overline{\mathsf{gts}}(R)(q)$, thus excluding a (hopefully) large set of principals that do not need to be involved in computation. We then proceed with computation of $\overline{\mathsf{gts}}(R)(q)$ by showing that the conditions of a general algorithmic convergence-theorem of Bertsekas [1] are satisfied, and hence we can appeal to previous results on the convergence of a certain totally asynchronous algorithm.

We present our problem in the more abstract setting of a distributed computation of the least fixed-point of a continuous endo-function on a cpo. We show that this indeed models our practical scenario (and of course, many others).

Abstract setting. We are given a cpo $(X, \sqsubseteq)$ of finite height h, and a natural number $n \in \mathbb{N}$. Writing $[n]$ for the set $\{1, 2, \ldots, n\}$, we have also a collection $C = (f_i : i \in [n])$ of n continuous functions, each of type $f_i : X^{[n]} \to X$. These functions induce a unique, continuous, global function $F = \langle f_i : i \in [n] \rangle : X^{[n]} \to X^{[n]}$ which has a unique least-fixed-point, $\mathsf{lfp}\ F \in X^{[n]}$. Define a dependency graph $G = ([n], E)$, where $[n]$ is the set of nodes, and the edges, given as a function $E : [n] \to \mathbf{2}^{[n]}$, model (possibly an over-approximation of) the dependencies of the functions in C, i.e., have $j \notin E(i)$ implies that function f_i does *not* depend on the value of "variable" j. We consider the nodes $[n]$ as network nodes that have memory and computational power. Each node $i \in [n]$ is associated with function f_i, and we assume that each node knows all nodes that it depends on, i.e., node i knows all edges $E(i)$.

Computational problem. Let $R \in [n]$ denote a designated node, called the *root*. The computational problem is for the root to compute the *local fixed-point value* $(\mathsf{lfp}\ F)_R$.

Concrete setting. We translate the trust-structure setting into our abstract setting by defining function f_R as policy π_R's entry for principal q. One then finds the dependencies of f_R by looking at which other policies this expression depends on. If f_R depends on entry w in π_z, then z is a node in the graph, and the function f_z is given by π_z's entry for w, with the dependencies of f_z given by the dependencies in the expression for w in π_z, and so on. From now on, we shall work in the abstract setting as it simplifies notation.

Note, that this translation might lead to a node z appearing several times in the dependency graph, e.g. with entries for principals w and y in π_z. We shall think of these as distinct nodes in the graph, although a concrete implementation would have node z play the role of two nodes, z_w and z_y. Note also, that the (minimal) dependency-graph is *not* modeling any network topology. Although the nodes of the graph represent concrete nodes in a physical communication-network, its edges do not represent any communication-links.

Communication model. We use an asynchronous communication-model, assuming no known bound on the time it takes for a sent message to arrive. We assume that communication is reliable in the sense that any message sent eventually arrives, exactly once, unchanged, to the right node, and that messages arrive in the order in which they are sent. We assume (in the spirit of the global-computing vision) an underlying physical communication-network allowing any node to send messages to any other node. Furthermore, we assume that all nodes are willing to participate, and that they do not fail. The assumptions of non-failure and correct order of delivery ease the exposition, but the fixed-point algorithm we apply is highly robust [1].

Our algorithm for fixed-point computation consists of two stages. In the first stage, the dependency graph $G = ([n], E)$ is distributedly computed so that each node knows the set of nodes that depend on it for the computation. In the second stage, this information is used in an asynchronous algorithm, performing the actual fixed-point computation.

2.1 Computing Trust-Dependencies

Setting up the dependency graph is very simple, and so we describe it only very briefly (see the full paper [17] for details). The goal of the dependency computation is for each node to obtain a list of the nodes that *depend on it* for the trust-value computation. For any node i, we denote the set $E(i)$ by i^+, and the set of nodes k for which $i \in E(k)$ (i.e. $E^{-1}(\{i\})$), by i^-. After the dependency computation, any node i knows i^+ and i^-. Node i will store i^+ and i^- in variables of the same name.

Computing the dependency graph reduces to a distributed reachability problem. Intuitively, one starts at the root, which sends a `mark` message to each of R^+, and, in turn, each node $i \in R^+$ reachable in one "step" from the root, will notify the nodes i^+ of this dependency, and so on (taking appropriate action when cycles are discovered). This can be implemented with high parallelism, and with the total number of messages sent $O(|E|)$, each message of bit length $O(1)$. Note, that we only "mark" the nodes that are reachable from R, which amounts to excluding any node that R does not depend on (directly or by transitivity) for computing its trust value for $q \in \mathcal{P}$.

2.2 An Asynchronous Algorithm

In this section, we assume that the dependency graph has already been computed. We show that we a now in a situation in which we can apply existing work of Bertsekas for computation of the least fixed-point. Bertsekas has a class of algorithms, called totally asynchronous (TA) distributed iterative fixed-point algorithms, and a general theorem which gives conditions ensuring that a specific TA algorithm will converge to the desired result. In our case, "converge to" means that each principal $i \in \mathcal{P}$ will compute a sequence of values $\bot_\sqsubseteq = i.t_0 \sqsubseteq i.t_1 \sqsubseteq \cdots \sqsubseteq i.t_k = (\mathsf{lfp}\, F)_i$. The general theorem is called the "Asynchronous Convergence Theorem" (ACT), and we use this name to refer to Proposition 6.2.1 of Bertsekas' book [1]. The ACT applies in any scenario in which the so-called "Synchronous Convergence Condition" and the "Box Condition" are satisfied. Intuitively, the synchronous convergence condition states that if the algorithm is executed synchronously, then one obtains the desired result. In our case, this amounts to requiring that the "synchronous" sequence $\bot_\sqsubseteq \sqsubseteq F(\bot_\sqsubseteq) \sqsubseteq \cdots$ converges to the least fixed-point, which is true. Intuitively, the box condition requires that one can split the set of possible values appearing during synchronous computation into a product ("box") of sets of values that appear locally at each node in the asynchronous computation. As a consequence of $\sqsubseteq$-monotonicity of the policies, the conditions of the Asynchronous Convergence Theorem are satisfied (the following Proposition 2.1), and so, we can deploy a TA distributed algorithm.

We now describe the algorithm and argue for its correctness. We will assume that each node i allocates variables $i.t_{cur}$ and $i.t_{old}$ of type X, which will later record the "current" value and the last computed value in X. Each node i has also an array, denoted by $i.m$. The array $i.m$ is of type X array, and will be indexed by the set i^+. Initially, $i.t_{cur} = i.t_{old} = \bot_\sqsubseteq$, and the array is also initialized with $\bot_\sqsubseteq$. For any nodes i and $j \in i^+$, when i receives a message from j (which is always a value $t \in X$), it stores this message in $i.m[j]$.

Asynchronous algorithm. Any node is always in one of two states: *sleep* or *wake*. All nodes start in the *wake* state, and if a node is in the *sleep* state, the reception of a message triggers a transition to the *wake* state. In the *wake* state any node i repeats the following: it starts by assigning to variable $i.t_{cur}$ the result of applying its function f_i to the values in $i.m$, i.e., node i executes assignment $i.t_{cur} \leftarrow f_i(i.m)$. If there is no change in the resulting value of $f_i(i.m)$ (compared to the last value computed, which is stored in $i.t_{old}$), it will go to the *sleep* state unless a message was received since $f_i(i.m)$ was computed. Otherwise, if a new value resulted from the computation (i.e., if $t_{old} \neq f_i(i.m)$), this value is sent to all nodes in i^-. Concurrently with this we can run a termination detection algorithm, which will detect when all nodes are in the *sleep*-state and no messages are in transit. Bertsekas has already addressed this problem with his termination-detection algorithm [1], which directly applies, yielding only a constant overhead in the message complexity.

To prove correctness of the asynchronous algorithm, we need only prove that the ACT is satisfied when all nodes initialize their trust-values ($i.m$ and $i.t_{old}$) to $\bot_\sqsubseteq$. However, we instead prove a slightly more general convergence-result which is useful when considering the interplay between the asynchronous-algorithm and policy-updates. The following concept of an *information approximation* is central.

Definition 2.1 (Information Approximation). *Let $F : X^{[n]} \to X^{[n]}$ be continuous. Say that a value $\bar{t} \in X^{[n]}$, is an* information approximation *for F if $\bar{t} \sqsubseteq \mathsf{lfp}\, F$ and $\bar{t} \sqsubseteq F(\bar{t})$.*

The following Proposition 2.1 shows that we can indeed appeal to the ACT.

Proposition 2.1 (Convergence Theorem). *Let $\bar{t}$ be any information approximation for F. Assume that after running the dependency-graph algorithm, the arrays of the nodes are initialized with $\bar{t}$. That is, for all nodes $i \in [n]$, and all $j \in i^+$ assume that $i.m[j] = \bar{t}_j$ and that $i.t_{old} = \bar{t}_i$. Then the synchronous convergence condition and the box condition of the asynchronous convergence theorem are both satisfied.*

To prove convergence of our algorithm, we simply invoke Proposition 2.1 in the case of the trivial information-approximation $\bar{t} = \bot_\sqsubseteq^n$. The asynchronous convergence theorem ensures that the asynchronous algorithm converges towards the right values at all nodes, and, because of our assumption of finite height cpos, the distributed system will eventually reach a state which is stable. In this state, each node i will have computed $(\mathsf{lfp}\, F)_i$.

Remarks. Since any node sends values only when a change occurs, by monotonicity of f_i, node i will send at most $h \cdot |i^-|$ messages, each of size $O(\log |X|)$ bits.[5] Node i will receive at most $h \cdot |i^+|$ messages, each message (possibly) triggering a computation of f_i. Globally, the number of messages is $O(h \cdot |E|)$ each of bit size $O(\log |X|)$. Hence, the communication complexity of our algorithm is linear in the height of the lattice used by the policies. An important global invariant in this algorithm is that any value computed locally at a node (by the assignment $i.t_{cur} \leftarrow f_i(i.m)$) is a component in an information approximation for F. That is, it holds everywhere, at any time, that (1) $i.t_{cur} \sqsubseteq (\mathsf{lfp}\, F)_i$ and (2) $i.t_{cur} \sqsubseteq f_i(i.m)$. To see this, note that $(1, 2)$ hold initially, and that both properties are preserved by the update $i.t_{cur} \leftarrow f_i(i.m)$ whenever $i.m[y] \sqsubseteq (\mathsf{lfp}\, F)_y$ for all $y \in i^+$ (which is always true). We state this fact as a lemma, as it becomes very useful in the next section where we consider fixed-point approximation-algorithms.

[5] In fact, there will be *only* $O(h)$ *different* messages, each sent to all of i^-. Consequently, a broadcast mechanism could implement the message delivery efficiently.

Lemma 2.1. *Any value $i.t_{cur} \in X$ computed by any node $i \in [n]$, at any time in the algorithm by the statement $i.t_{cur} \leftarrow f_i(i.m)$, is a part of an information approximation for F, in the sense that $i.t_{cur} \sqsubseteq (\mathsf{lfp}\, F)_i$ and $i.t_{old} \sqsubseteq i.t_{cur}$.*

3 Approximation techniques

In this section, we present two techniques for safe and efficient approximation of the fixed-point. Consider a situation in which a client principal p wants to access a resource controlled by server v. Assume that the access-control policy of v is that, to allow access, its trust in p should be trust-wise above some threshold $t_0 \in X$, i.e., the fixed-point should satisfy $t_0 \preceq (\mathsf{lfp}\, \Pi_\lambda)(v)(p)$. The goal of the approximation techniques is to allow the server to (soundly) make its security decision without having to actually compute the exact fixed-point value. Instead, the server is able to efficiently compute an approximating global trust-state $\bar{p} : \mathcal{P} \to \mathcal{P} \to X$ which is related to the fixed point in such a way that the desired property can be asserted.

We need some preliminary terminology. Let $T = (X, \preceq, \sqsubseteq)$ be a trust structure, i.e. $(X, \sqsubseteq)$ is a cpo with bottom $\perp_\sqsubseteq$ and $(X, \preceq)$ is a partial order (not necessarily complete). We assume also that $(X, \preceq)$ has a least element, denoted $\perp_\preceq$. If for any countable $\sqsubseteq$-chain $C = \{x_i \in X \mid i \in \mathbb{N}\}$ and any $x \in X$ we have (i) $x \preceq C$ implies $x \preceq \bigsqcup C$ and (ii) $C \preceq x$ implies $\bigsqcup C \preceq x$, then $\preceq$ can be said to be *$\sqsubseteq$-continuous*.

3.1 Bounding "Bad Behaviour"

This first technique lets a client convince a server that its trust in the client is (trust-wise) above a certain level. The technique is based on the following proposition.

Proposition 3.1. *Let $(X, \preceq, \sqsubseteq)$ be a trust structure in which $\preceq$ is $\sqsubseteq$-continuous. Let $\bar{p} \in X^{[n]}$, and $F : X^{[n]} \to X^{[n]}$ be any function that is $\sqsubseteq$-continuous and $\preceq$-monotonic. If we have $\bar{p} \preceq (\lambda k \in [n].\perp_\sqsubseteq)$ and $\bar{p} \preceq F(\bar{p})$, then $\bar{p} \preceq \mathsf{lfp}_\sqsubseteq F$.*

Note that the conclusion of the proposition is an assertion that is useful for authorization; if the server knows a $\bar{p} \in X^{[n]}$ which is sufficient to allow an authorization, and knows also that $\bar{p} \preceq \mathsf{lfp}_\sqsubseteq F$, then since the ideal global trust-state is *trust-wise* above $\bar{p}$, then it is a sound decision to allow the authorization. This idea is the basis of an efficient protocol for a kind of "proof-carrying" authorization, similar to the traditional notion of proof-of-compliance, used e.g. in PolicyMaker [5].

Consider for simplicity the "MN" trust-structure T_{MN} from Section 1, which satisfies the information-continuity requirement. Recall that, in this structure, trust values are pairs (m, n) of natural numbers, representing $m + n$ past interactions; m of which where classified 'good', and n, classified as 'bad'.[6] The orderings are given by $(m, n) \sqsubseteq (m', n') \iff m \leq m'$ and $n \leq n'$, and $(m, n) \preceq (m', n') \iff m \leq m'$ and $n \geq n'$.

Suppose principal p wants to efficiently convince principal v, that v's trust value for p is a pair (m, n) with the property that n is less than some fixed bound $N \in \mathbb{N}$ (i.e., giving v an upper bound on the amount of recorded "bad behaviour" of p). Let us assume that v's trust policy π_v is monotonic, also with respect to $\preceq$, and that it depends on a large set S of principals. Assume also that it is sufficient that principals a and b in S have a reasonably "good" trust-value for p, to ensure that v's trust-value for p is not too "bad". An example policy with this property could be written in the language of Carbone *et al.* [8] as

$$\pi_v \equiv \lambda x : \mathcal{P}.(\ulcorner a \urcorner(x) \wedge \ulcorner b \urcorner(x)) \vee \bigwedge_{s \in S \setminus \{a,b\}} \ulcorner s \urcorner(x)$$

The construct $\ulcorner \cdot \urcorner$ represents *policy reference* or *delegation*, e.g., if a and x are principal identities then expression $\ulcorner a \urcorner(x)$ "evaluates" to the value that a's trust policy specifies for x. The construct $e \vee e'$ represents least upper-bound in the trust-ordering (intuitively, "trust-wise maximum" of e and e'), and similarly $\wedge$ represents greatest lower-bound ("trust-wise minimum").[7] Thus, informally, the above policy says that any principal p should have "high trust" with a and b, or, with *all of* $s \in S \setminus \{a, b\}$, for the v to assign "high trust" to p. Now, if p knows that it has previously performed well with a and b, and knows also that v depends on a and b in this way, it can engage in the following protocol.

Protocol. Principal p sends to v the "trust-state" $t = [(v, p) \mapsto (0, N), (a, p) \mapsto (0, N_a), (b, p) \mapsto (0, N_b)]$ which can be thought of as a "proof" (analogous to a 'proof-of-compliance') or a "claim" made by p, stating that $(0, N) \preceq (\mathsf{lfp}\, \Pi_\lambda)(v)(p)$ (and similarly for a and b). Upon reception, v first extends t to a global trust state, which is the extension of t to a function $\bar{p}$ of type $\mathcal{P} \to \mathcal{P} \to T_{MN}$, given by

$$\bar{p} = \lambda x \in \mathcal{P} \lambda y \in \mathcal{P}. \begin{cases} (0, N) & \text{if } x = v \text{ and } y = p \\ (0, N_a) & \text{if } x = a \text{ and } y = p \\ (0, N_b) & \text{if } x = b \text{ and } y = p \\ (0, \infty) & \text{otherwise} \end{cases}$$

To check the proof, principal v must verify that $\bar{p}$ satisfies the conditions of Proposition 3.1. First, v must check that

[6]To be precise, the set $\mathbb{N}^2$ is completed by allowing also value ∞ as "m" or "n" or both.

[7]The example policy assumes that $(X, \preceq)$ is a lattice, meaning that for any $x, y \in X$ both $x \vee y$ and $x \wedge y$ exist. Furthermore operations $\vee$ and $\wedge$ must be continuous *also with respect to the information ordering*. In many trust-structure this is often the case [8].

$\bar{p}(x)(y) \preceq \bot_{\sqsubseteq} = (0,0)$ for all x, y. But this holds trivially if $y \neq p$ or $x \neq v, a, b$ because then $\bar{p}(x)(y) = (0,\infty) = \bot_{\preceq}$. For the other few entries it is simply an order-theoretic comparison $\bar{p}(x)(y) \preceq (0,0)$. Now v tries to verify that $\bar{p} \preceq \Pi_\lambda(\bar{p})$. To do this, v verifies that $(0,N) \preceq \pi_v(\bar{p})(p)$. If this holds then v sends the value t to a and b, and ask a and b to perform a similar verification (e.g. $(0,N_a) \preceq \pi_a(\bar{p})(p)$). Then a and b reply with 'yes' if this holds and 'no' otherwise. If both a and b reply 'yes', then p is sure that $\bar{p} \preceq \Pi(\bar{p})$: by the checks made by v, a and b, we have that $\bar{p}(x)(y) \preceq \Pi_\lambda(\bar{p})(x)(y)$ holds for pairs $(x,y) = (v,p), (a,p), (b,p)$, but for all other pairs it holds trivially since $\bar{p}$ is the $\preceq$-bottom on these. By Proposition 3.1, we have $\bar{p} \preceq \mathsf{lfp}\, \Pi_\lambda$, and so, v is *ensured* that its trust value for p is $\preceq$-greater than $(0,N)$.

We have illustrated the main idea of the protocol by way of an example, but the general technique for verifying a proof should be clear. In general, the proof $\bar{p}$ may include a larger number of principals, which would then have to be involved in the verification process.

Remarks. Our approximation protocol has very much the flavour of a proof-carrying authorization: the requester (or prover) must provide a proof that its request should be granted. It is then the job of the service-provider (or verifier) to check that the proof is correct. The strength of this protocol lies in replacing an entire fixed-point computation with a few local checks made by the verifier, together with a few checks made by a subset of the principals that the verifier depends on. An interesting property of this protocol is that part of the information that the prover needs to supply should already be known to the prover; it should already know who it has performed well with in the past (e.g. in our example above, p could know the bounds N_a and N_b because of its previous interaction with a and b). There are, however, two important restrictions to this approach. First, as in the example, in order to construct its proof, the prover needs information about the verifiers trust policy and of the policies of those whom the verifier depends on. If policies are secret, it is not clear how the verifier would construct this proof. Second, because of the requirement in Proposition 3.1 that $\bar{p} \preceq \bot_{\sqsubseteq}$, the protocol can usually only be used to prove properties stating "not too much bad behaviour," and *not* properties guaranteeing sufficiently "good" behaviour.

Notice that the protocol for exploiting Proposition 3.1 has a message complexity which is independent of the height of the cpo; in particular, it works also for infinite height cpos. In contrast, the algorithm for computing fixed-points has message complexity $O(h \cdot |E|)$.

We present now another approach which requires more computation and communication, but does not have the two mentioned restrictions.

3.2 Exploiting Information Approximations

The approximation technique developed in this section is different from that of the "proof-carrying" protocol in the previous section. In this section, we not require the "prover" (client) to provide any information. Instead, we derive an approximation from a "snapshot" of the state of the asynchronous fixed-point algorithm from Section 2.2. The "verifiers" (servers) are then able make a collection of local checks on this snapshot, allowing them to infer that the fixed-point value must be trust-wise above the snapshot-value. The technique is based on the following proposition.

Proposition 3.2. *Let $(X, \preceq, \sqsubseteq)$ be a trust structure in which $\preceq$ is $\sqsubseteq$-continuous. Let $\bar{t} \in X^{[n]}$, and $F : X^{[n]} \to X^{[n]}$ be any function that is $\sqsubseteq$-continuous and $\preceq$-monotonic. Assume that $\bar{t}$ is an information approximation for F. If $\bar{t} \preceq F(\bar{t})$ then $\bar{t} \preceq \mathsf{lfp}\, F$.*

This proposition is very useful because, by Lemma 2.1, a global invariant in the asynchronous fixed-point algorithm is that all values computed are information approximations for F. This means that we can combine the algorithm with a protocol that, intuitively, implements the check for the condition $\bar{t} \preceq F(\bar{t})$ in the above proposition.

Imagine that during the execution of the asynchronous algorithm, there is a point in time, in which no messages are in transit, all nodes i have computed their function f_i, and sent the value $f_i(i.m)$ to all that depend on it. Thus we have a "consistent" state in the sense that for any node x and any node $y \in x^+$ we have $x.m[y] = y.t_{cur}$. In particular if x and z both depend on y, then they agree on y's value: $x.m[y] = y.t_{cur} = z.m[y]$. In this ideal state, there is a consistent vector $\bar{t}$ which, by Lemma 2.1, is an information approximation for F, i.e. $\bar{t}$ contains the values $\bar{t}_i = i.t_{cur}$ for nodes $i \in [n]$. If the state of the distributed system was frozen at this point, and all nodes x, simultaneously make the check $x.t_{cur} \preceq f_x(x.m)$, then vector $\bar{t}$ satisfies $\bar{t} \preceq F(\bar{t})$. Since $\bar{t}$ is an information approximation for F, by Proposition 3.2, the root node R knows that $\bar{t}_R \preceq \mathsf{lfp}\ F_R$, which is what we want.

Of course, the ideal situation described above would rarely occur in a real execution. The aim of the approximation technique in this section, is to enforce, during execution of the asynchronous algorithm, a consistent view of such an ideal situation. In so-called snapshot-algorithms (see Bertsekas [1]), the (local views of the) global state of the system is recorded during execution of an algorithm. Our problem is slightly less complicated since we are not interested in the status of communication links, but slightly more complicated since each snapshot-value must be propagated to a specific set of nodes. The full paper describes a snapshot-algorithm implementing the above idea. The algorithm sends a constant number of messages for each edge in G, hence its message complexity is $O(|E|)$.

Interestingly, it turns out that the two propositions of this section are actually instances of a more general theorem, which gives rise to a generalized approximation-protocol, that can be seen as a combination of the two techniques presented in this section. Due to space restrictions, we leave this to the full paper [17].

We note finally that the $\sqsubseteq$-continuity property, required of $\preceq$ in our propositions, is satisfied for all interesting trust-structures we are aware of: Theorem 3 of Carbone *et al.* [8] implies that the information-continuity condition is satisfied for all interval-constructed structures. Furthermore, their Theorem 1 ensures that interval-constructed structures are complete lattices with respect to $\preceq$ (thus ensuring existence of $\bot_{\preceq}$). Several natural examples of non-interval domains can also be seen to have the required properties [15]. The requirement that all policies π_p are monotonic also with respect to $\preceq$ is not unrealistic. Intuitively, it amounts to saying that if everyone raises their trust-levels in everyone, then policies should not assign lower trust levels to anyone.

4 Conclusion

We have presented distributed algorithmic techniques for approximation of the least fixed-point of a collection of continuous functions, focusing particularly on 'trust structures' – sets partially ordered by two relations: the information ordering and the trust ordering. When the functions are trust policies that are monotonic with respect to the trust-ordering, and, if the trust ordering is continuous with respect to the information ordering, we have illustrated two protocols allowing principals to soundly reason about the fixed-point values without having to compute the exact fixed-point.

Our work is based solely on theoretical analyses, but these lead to promising conclusions regarding the usefulness of the trust-structure framework, justifying further validation via actual implementation. The trust-structure framework has a concrete instance in the SECURE project [6, 7] which deploys a specific class of trust structures, using probabilistic information in its modeling of trust [15, 20]. As part of this project, we are planning on developing prototype implementations of the techniques in this paper.

Apart from the application in implementing trust-structure-based systems, the technique for fixed-point computation is general enough to be used in any cpo with bottom (or complete lattice). In particular, the techniques could be the basis of a distributed implementation of a variant of Weeks' model of trust-management systems [24], in which credentials could be stored by the issuing authorities instead of being presented by clients. This would support revocation, implemented simply as a trust-policy update at the authority revoking the credential.

Future work. We have considered the dependency graph induced by a collection of policy functions. Since this graph is not necessarily equal to the physical communication graph, the algorithms may have to send messages over several links in order to represent the sending of a message over a single edge in the dependency graph. It would be a relevant and interesting topic to consider to what extent the quality of the embedding affects the convergence rate of the fixed-point algorithm.

We have analyzed the worst-case message-complexities of our algorithms. It could also be interesting to try and analyze the "amortized" complexity of our algorithms. For example, if principal R wants to know its trust in q, it can run the algorithm presented in this paper to compute or approximate this value. Now, after some time has passed, principals might have made additional observations about q. Suppose that, at some point later, R wants to compute its trust in q. Since principals reuse the information gained from the last computation, the second computation would be significantly faster.

Related Work. As mentioned previously, Weeks has developed a mathematical framework [24] suitable for modeling many traditional trust-management systems (e.g. [4, 10, 3, 11, 12]). The framework is based on defining a global trust-state ("authorization map" [24]) by existence of least fixed-points of monotonic endo-functions on complete lattices. The trust-structure framework [8, 19], introduces a notion of *information* into the framework of Weeks. The primary difference between the two frameworks is that, in trust structures, least fixed-points are with respect to *information*, whereas in Weeks' framework they are with respect to *trust* (indeed, there is no notion of 'information ordering', and 'trust' is identified with authorization [24]). Another important difference is that in Weeks' framework, the trust policies (licenses) are carried by clients instead of being stored at the issuing servers. This means that the operational approach is to let clients present, along with their request, a set of licenses, which, together, give rise to what corresponds to function Π_λ. It is now the job of the server to (locally) compute the fixed-point, and decide how to respond. In contrast, in the trust-structure framework, the trust policies are naturally distributed. Each principal p, autonomously controls and stores its policy, π_p. This leads naturally to a distributed approach to computation of fixed-points.

The idea of computing *local* fixed-points has been recognized also by Vergauwen *et al.* in the non-distributed context of static program-analysis [23]. Dimitri Bertsekas has developed a substantial body of work on distributed and parallel algorithms for fixed points, and this paper applies his asynchronous convergence theorem [1] to prove correctness of a distributed fixed-point algorithm. Finally, the Eigen-

Trust system also defines its global trust-state by existence of unique (non order-theoretic) fixed-points [14], and the basic EigenTrust algorithm is essentially Bertsekas' globally synchronous algorithm.

Acknowledgments. We thank everyone in the SECURE project consortium for their cooperation; Mogens Nielsen in particular, for the initial developments of 'proof-carrying authorization'. We thank also Vladimiro Sassone and John Mitchell for their help in making this a better paper.

References

[1] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall International Editions. Prentice-Hall, Inc., 1989.

[2] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. D. Keromytis. The role of trust management in distributed systems security. In J. Vitek and C. D. Jensen, editors, *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, volume 1603 of *Lecture Notes in Computer Science*, pages 185–210. Springer, 1999.

[3] M. Blaze, J. Feigenbaum, and A. D. Keromytis. KeyNote: Trust management for public-key infrastructures. In *Proceedings from Security Protocols: 6th International Workshop, Cambridge, UK, April 1998*, volume 1550, pages 59–63, 1999.

[4] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *Proceedings from the 17th Symposium on Security and Privacy*, pages 164–173. IEEE Computer Society Press, 1996.

[5] M. Blaze, J. Feigenbaum, and M. Strauss. Compliance checking in the policymaker trust management system. In *Proceedings from Financial Cryptography: Second International Conference (FC'98), Anguilla, British West Indies, February 1998*, pages 254–274, 1998.

[6] V. Cahill and E. Gray *et al.* Using trust for secure collaboration in uncertain environments. *IEEE Pervasive Computing*, 2(3):52–61, 2003.

[7] V. Cahill and J.-M. Seigneur. The SECURE website. `http://secure.dsg.cs.tcd.ie`, 2004.

[8] M. Carbone, M. Nielsen, and V. Sassone. A formal model for trust in dynamic networks. In *Proceedings from Software Engineering and Formal Methods (SEFM'03)*. IEEE Computer Society Press, 2003.

[9] A. Chander, D. Dean, and J. Mitchell. Reconstructing trust management. *Journal of Computer Security*, 12(1):131–164, 2004.

[10] Y.-H. Chu, J. Feigenbaum, B. LaMacchia, P. Resnick, and M. Strauss. REFEREE: Trust management for (web) applications. *Computer Networks and ISDN Systems*, 29(8–13):953–964, 1997.

[11] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomsa, and T. Ylonen. SPKI Certificate Theory. RFC 2693, ftp-site: `ftp://ftp.rfc-editor.org/in-notes/rfc2693.txt`, September 1999.

[12] T. Jim. SD3: A trust management system with certified evaluation. In *Proceedings from the 2001 IEEE Symposium on Security and Privacy, Oakland, California*, pages 106–116. IEEE Computer Society Press, 2001.

[13] A. Jøsang, R. Ismail, and C. Boyd. A survey of trust and reputation for online service provision. Decision Support Systems, (to appear, preprint available online: `http://security.dstc.edu.au/staff/ajosang`), 2004.

[14] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in P2P networks. In *Proceedings from the twelfth international conference on World Wide Web, Budapest, Hungary*, pages 640–651. ACM Press, 2003.

[15] K. Krukow. On foundations for dynamic trust management. Unpublished PhD Progress Report, available online: `http://www.brics.dk/~krukow`, 2004.

[16] K. Krukow, M. Nielsen, and V. Sassone. A formal framework for concrete reputation-systems with applications to history-based access control. Submitted. Available online: `http://www.brics.dk/~krukow`, 2005.

[17] K. Krukow and A. Twigg. Distributed approximation of fixed-points in trust structures. Technical Report RS-05-6, BRICS, University of Aarhus, Jan. 2005. Available online: `http://www.brics.dk/RS/05/6`.

[18] N. Li and J. Mitchell. A role-based trust-management framework. In *Proceedings from DARPA Information Survivability Conference and Exposition (DISCEX III)*, pages 201–213. IEEE Computer Society Press, 2003. (vol. 1).

[19] M. Nielsen and K. Krukow. Towards a formal notion of trust. In *Proceedings from the 5th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming (PPDP'03)*, pages 4–7. ACM Press, 2003.

[20] M. Nielsen and K. Krukow. On the formal modelling of trust in reputation-based systems. In J. Karhumäki, H. Maurer, G. Paun, and G. Rozenberg, editors, *Theory Is Forever: Essays Dedicated to Arto Salomaa*, volume 3113 of *Lecture Notes in Computer Science*, pages 192–204. Springer Verlag, 2004.

[21] PGPi website. An introduction to cryptography, in pgp user's guide 7.0. ftp: `ftp://ftp.pgpi.org/pub/pgp/7.0/docs/english/IntroToCrypto.pdf` website: `http://www.pgpi.org/doc`, 2000.

[22] V. Shmatikov and C. Talcott. Reputation-based trust management. *Journal of Computer Security*, 13(1):167–190, 2005.

[23] B. Vergauwen, J. Wauman, and J. Lewi. Efficient fixpoint computation. In B. Le Charlier, editor, *Proceedings from Static Analysis (SAS'94)*, pages 314–328. Springer, Berlin, Heidelberg, 1994.

[24] S. Weeks. Understanding trust management systems. In *Proceedings from the 2001 IEEE Symposium on Security and Privacy*, pages 94–106. IEEE Computer Society Press, 2001.

[25] G. Winskel. *Formal Semantics of Programming Languages : an introduction*. Foundations of computing. The MIT Press, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1993.