USING RECOMMENDATIONS FOR MANAGING TRUST IN DISTRIBUTED SYSTEMS

Alfarez Abdul-Rahman & Stephen Hailes

{F.AbdulRahman,S.Hailes}@cs.ucl.ac.uk
Department of Computer Science, University College London,
Gower Street, London WC1E 6BT, United Kingdom.

Abstract

Each time we carry out vital communication in any distributed computer system such as the Internet, we face an inherent risk. This risk arises because we can never be completely certain about the trustworthiness of entities that mediate our on-line interactions. To minimise this risk, users must be given the chance to assess trust on the network, and be given an opportunity to pick an option with the least level of perceived risk. In this paper, we explain why traditional network security mechanisms are incomplete in their function to manage trust, and provide a general model based on recommendations.

1. INTRODUCTION

Trust is a notion central to secure distributed systems communications and transactions. Within this context, when something is proven to be secure, it is 'trusted'. Well known techniques to ensure that something is 'trusted' have been developed and strengthened, and these techniques include cryptographic algorithms for secrecy and digital signatures, authentication protocols for proving authenticity, and access control methods for managing authorisation.

However, these techniques do not say much about the wider notion of an entity's 'trustworthiness'. For example, cryptographic algorithms cannot say if a piece of digitally signed code has been authored by competent programmers and a signed public-key certificate does not tell you if the owner is an industrial spy.

With the increasing pervasiveness of the Internet, and examples of its use like downloading software and on-line commerce become commonplace, trust management becomes an increasingly essential component. In [Luh79], Niklas Luhmann said that if we live without trust, then human interaction will not progress beyond the very trivial, as only limited forms of action and cooperation is possible. Acting on behalf of a human being, any computer is merely an extension of a person, and therefore must be given the same capability as humans to manage and reason about trust. This capability is currently lacking, so a tool for managing trust, which will complement current security technology, must be designed.

In this paper, we present our approach to the problem of trust management, which uses *recommendations*.

We will continue by clarifying our concept of trust in §2. In §3, we will look at current practices in security, and discuss their suitability to trust management. Then in §4, we provide an outline of our approach, and justification of our design decisions. A *trust model* is proposed in §5, and details of a *Recommendation protocol* are fleshed out in §6. In §7, we show how trust is calculated, followed by a discussion of issues that relate to our work, in §8. Finally, we conclude in §10.

2. DEFINITION OF TRUST

In this paper, we use Gambetta's definition of trust [Gam90]:

"trust (or, symmetrically, distrust) is a particular level of the subjective probability with which an agent will perform a particular action, both before [we] can monitor such action (or independently of his capacity of ever to be able to monitor it) and in a context in which it affects [our] own action".

Of importance here are three points made in the definition above: 1) that trust is subjective, 2) trust affects those actions that we cannot monitor, and 3) the level of trust depends on how our own actions are in turn affected by the agent's actions.

3. MOTIVATION

In the previous section, we gave brief examples illustrating the need for more effective trust management techniques. We now discuss in more detail the properties of current security practices, and its issues which motivate a need for complementary trust management schemes.

The following trends in current security practice impact the *management of trust*:

a) Hard security

b) Centralised protocols

c) Implicit trust assumptions

These approaches have, and will continue to have, important roles to play in the security of network formation systems. However, they are incomplete as a general tool for managing trust. Below, we highlight the shortcomings of points a) – c) above, with respect to trust management.

3.1 Hard security

Currently, most security solutions can be categorised as 'hard' mechanisms. These mechanisms have an 'all or nothing' property, i.e. access is granted fully, or not at all. Cryptographic algorithms and firewalls are examples of hard security mechanisms.

Hard security mechanisms do not say anything about trust[1].

[1] "[A] program's hostility cannot be decided by any level of

Cryptography, for instance, is essential in ensuring the integrity and privacy of statements about trust, but it is not a mechanism for managing trust in itself.

Hard techniques work for applications where complete certainty is attainable, e.g. whether Alice *knows* the key K or not. As the notion of trust precludes some element of uncertainty, an alternative 'soft' approach will be more suitable for trust management. 'Soft security' is the term used by Rasmusson et al [RJ96,RRJ96] to describe a 'social control' model which acknowledges that malicious entities may exist among benign ones.

3.2 Centralised protocols

Centralised protocols are protocols which use a 'well-known' common trusted intermediary, call it the trusted authority (TA), to form a trust relationship between two mutually distrusting entities. This need to refer to a centrally trusted point poses a major problem as a *general* trust framework, because the assumed *objectivity* of trust will not work. In our definition of trust, trust is *subjective*.

Furthermore, a TA can never be a good enough recommender for *everyone* in a large distributed system. Its credibility depletes, and its recommendations increase in *uncertainty*, as its community of trustees grows.

A decentralised approach to trust management makes sense, simply because trust decisions cannot be forced onto entities – they make up (or must be allowed to make up) their own minds. In the words of Phil Zimmermann: "Centralised trust goes against the grain of human interaction"[2].

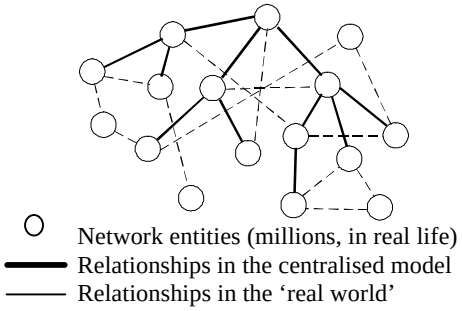

Figure 1 Centralisation vs. the decentralised 'real world'.

3.3 Implicit trust assumptions

Assumptions are always made about which entities can be trusted in a system. In common practice, the 'trusted' label is given to products or systems that has undergone a rigorous set of tests, and proven to have met certain criteria.

However, the 'trusted' label is misleading. 'Trusted' generally implies that 'nothing can go wrong', which implies that the tests covered all eventualities. Surely, this cannot be possible! This is a flaw of the Trusted Computing Base argument [DoD85] as highlighted by panel discussions in [Zur97], and also the underlying message in Bruce Schneier's words: "No amount of general beta testing will reveal a security flaw, and there's no test possible that can prove the absence of flaws" [Sch97].

If a secure system is desired, trust assumptions must be *explicit*. It is insufficient to say that Alice trusts Bob. More qualification is required: What does Alice trust Bob for, how much does Alice trust Bob and under what circumstances does that trust relationship hold or break?

3.4 Transitivity of trust

Most authentication protocols rely on the basic assumption that trust is *transitive*, i.e. if Alice trusts Bob, and Bob trusts Cathy, then it follows that Alice trusts Cathy. This, however, is an erroneous assumption. *Trust is not transitive* [Jøs96]. If Alice trusts Bob, and Bob trusts Cathy, it does not follow that Alice trusts Cathy. Alice's trust in Bob has no relation whatsoever to Bob's trust in Cathy.

The *decision* on whether to trust or distrust a subject, or a given authentication chain is ultimately made by the entity, and cannot be automated by any protocol.

4. OUR PROPOSAL

Our proposal extends and generalises current approaches to security and trust management, based upon four goals:

1. To adopt a *decentralised* approach to trust management.
2. To *generalise* the notion of trust.
3. To lessen ambiguity by making trust statements more *explicit*.
4. To facilitate the exchange of trust-related information via a *common protocol*.

We provide further justification for these goals below.

4.1 Decentralisation

As discussed earlier, trust management requires a decentralised approach, which will complement centralised approaches to give a more complete general trust management framework.

With decentralisation, each rational entity will be *allowed* to take responsibility for its own fate. This is a basic human right[3]. His policies need not be communicated, so there is less ambiguity and no effort involved in trying to understand them. Each entity then makes decisions for itself, on its own policies.

The disadvantage of decentralisation is that more responsibility and expertise is required on the user for managing trust policies. However, entities on the network will still have the option of relying on centralised trust models so that this responsibility can be assigned to their trusted authority, if they wish to do so. Decentralisation does not completely replace current centralised approaches, but it gives entities a choice of managing their own trust.

4.2 Generalising trust

Trust involves many aspects of an entity. When we say we trust someone, we know, with a large amount of certainty,

cryptography" [RRJ96].

[2] See [Abd97].

[3] See the UN Declaration of Human Rights.

exactly which *aspects* of trust we are referring to. For example, we trust that our car mechanic will carry out a satisfactory job of *car servicing*, but not to also handle our domestic *plumbing* needs. There are also instances when we trust one entity more than another, e.g. we trust one car mechanic more than another for some reason. This hints at different *levels* of trust.

To be able to capture this potentially large amount trust information, we need to *generalise* trust information. In our model, we have done this by using two types of trust information; *trust categories* to represent which aspect of trust one is referring to, and *trust values* for the different levels of trust within each category.

4.3 Explicit trust statements

The reason for making trust explicit is straightforward, i.e. to lessen ambiguity in recommendations which contain trust statements. The issues relating to implicit trust assumptions is discussed in §3.3. In our model, we have introduced *trust categories* and *trust values* to make trust statements more explicit.

4.4 A common protocol

We have proposed a *Recommendation Protocol* to facilitate the propagation of trust information. A protocol is essential as a standard vehicle for the exchange of trust information, and to avoid ambiguities in queries or requests for recommendation.

We argue that it makes sense to *recommend* trust because in the absence of an infinite pool of resources, entities, just as humans do, rely on information from others. We also stress that this protocol does not assume that trust is transitive, merely that agents are allowed to use trust-related *recommendations* from recommenders that *they trust*.

4.5 Novelty and suitability of proposed approach

Our approach is intended to complement current security practices by forming a general model within which trust can be more effectively managed. This is not 'yet another certification mechanism`. In a world where people live with *uncertainty*, our model copes with these uncertainties by allowing entities to reason with different degrees of trust, rather than just complete or incomplete trust.

Our model is concerned with the *general notion of trust*, one that goes beyond cryptographic protocols. This is important because entities need a flexible means to ascertain a variety of properties about a variety of other entities. For this to work we need to generalise trust.

We believe that our model will be most suited to trust relationships that are less formal, temporary or short-term trust relationships or ad-hoc commercial transactions. Our model will not be suited to formal trust relationships based on legally binding contracts.

5. THE TRUST MODEL

In this section, we explain how trust is defined in our trust model by describing its elements. This section can also be regarded as containing the assumptions that we have made in designing our trust model.

5.1 Agents

Entities that are able to execute the Recommendation Protocol are called *agents*. This is to differentiate from static entities like printers and disk volumes. Agents may recommend any entity, but only agents can make and receive recommendations.

5.2 Trust relationships

A trust relationship exists between A and B when A holds a belief about B's trustworthiness. However, the same belief in the reverse direction need not exist at the same time. In other words, A trust relationship is unidirectional.

The properties of a trust relationship in our model are:

1. It is always between exactly two entities.
2. It is non-symmetrical.
3. It is non-transitive.

If mutual trust exists between the same entities, we represent them as two separate trust relationships. This allows each of these relationships to be manipulated independently.

Two different types of relationships are distinguished.

If Alice trusts Bob, then there is a *direct trust relationship*. If Alice trusts Bob to give recommendations about other entities' trustworthiness, then there is a *recommender trust relationship* between Alice and Bob.

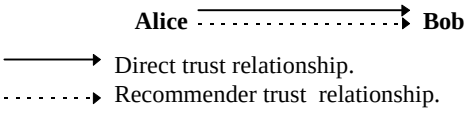

Figure 2 Recommender trust: Alice trusts Bob to recommend other entities

Trust relationships exists only within each entity's own database. Therefore, there is no such thing as a 'global map' of trust relationships in our model. This also makes trust relationships in our model highly volatile. The ability for each entity to revise the properties of each relationship at any time also makes trust relationships unstable.

By relaxing the constraints on *how* to build trust relationships, we are able to allow this model to be used for any type of trust architecture[4], e.g. hierarchical, digraphs or hybrids. Most architectures are policy driven, i.e. the shape of the architecture reflects the policies used to build them. Since we do not incorporate policies in our model, it is open to arbitrary architectures[5].

5.3 Trust Categories

Agents use *Trust Categories* to express trust towards other agents in different ways depending upon which particular

[4] The combined graph of trust relationships.
[5] See [AH97] for examples.

characteristic or aspect of that entity is under consideration at that moment. For example, we trust a CA to certify public keys (category "sign-key"), but not to attest to the key-holder's credit status (category Credit").

5.4 Trust Values

Trust values are used to represent the different levels of trust an entity may have in the other.

Naturally, there is no one universal value system because its use is application specific. However, standardisation is important for interoperability. Therefore, is important that a value system is proposed, even if we must base its semantics on pure intuition. Below, we outline the trusts values and their meaning as used in our trust model[6].

Trust values in our model are constrained within each category and arrre independent of values in other categories.

Two types of values are used, and the types of trust relationships they are relevant to as described in §5.2:

1. *Direct trust value*: This is relevant to *direct trust relationships*.
2. *Recommender Trust Value.* This is relevant to *recommender trust relationships.*

The values and their descriptions are given below.

Value	*Meaning*	*Description*
-1	Distrust	Completely untrustworthy.
0	Ignorance	Cannot make trust-related judgement about entity.
1	Minimal	Lowest possible trust.
2	Average	Mean trustworthiness. Most entities I know of have this trust level.
3	Good	More trustworthy than most entities.
4	Complete	Completely trust this entity.

Table 1 Direct Trust Value Semantics

Value	*Meaning*	*Description*
-1	Distrust	Completely untrustworthy.
0	Ignorance	Cannot make trust-related judgement about agent.
1 2 3 4	'Closeness' of recommender's judgement to own judgement about trustworthiness. See example below.	

Table 2 Recommender Trust ValueSemantics

5.5 Reputation and Recommendation

The concatenation of an entity's Id or name, the trust category and a trust value is called a *Reputation.*

A *Recommendation* is a communicated trust information, which contains *Reputations.*

[6] Other examples of trust values are in [Maurer, Reiter, BBK]

Each agent stores reputation records in their own private databases and uses this information to make recommendations to other agents.

5.6 Summary of definitions

Entity*:* any object in a network.

Agent*:* any entity that is capable of making trust-related decisions (therefore able to participate in the Recommendation Protocol).

Direct trust*:* trust in an entity, within a specific category and with a specific value.

Recommender trust*:* trust in an agent to recommend other entities.

Trust Category*:* the specific aspect of trust relevant to a trust relationship.

Trust Value*:* the amount of trust, within a trust category, in a trust relationship.

Reputation*:* trust information which contains the name of the entity, the trust category, and a trust value.

Recommendation*:* reputation information that is being communicated between two agents about another entity.

6. RECOMMENDATION PROTOCOL

Before going into detail, we stress that *recommended trust* is different from *transitive trust*, in that in recommended trust, we use the recommendation to assist us in forming trust *opinions* about others, and not to concretely use that information to automatically build trust relationships, as transitivity suggests. Furthermore, for brevity and clarity, we will leave out details on message integrity and privacy issues and concentrate on the trust-related content of the recommendation protocol messages.

To recap, each agent may be a *recommender*, or a *requestor* of a recommendation. Any entity may be a *target* for a recommendation.

6.1 Message structure

A requestor issues a recommendation request message, or an `RRQ`, and receives a `Recommendation` message. Recommendations can be refreshed or revoked using the `Refresh` message. These messages have the following structure:

6.1.1 RRQ

```
RRQ ::= Requestor_ID, Request_ID,
     Target_ID, Categories, GetPKC,
     Expiry

Categories ::= SET OF {Category_Name}
```

6.1.2 Recommendation

```
Recommendation ::= Requestor_ID,
     Request_ID, Target_ID, Return_Path,
     [Recommendation_slip, TargetPKC |
     NULL ]

Return_Path ::= SEQUENCE OF
```

```
     {Recommender_ID}

Recommendation_slip ::= SET OF SEQUENCE
     {Category_Name, Trust_value,
     Expiry}
```

6.1.3 Refresh

```
Refresh ::= Target_ID,
     Recommendation_chain,
     Recommendation_slip
```

`Requestor_ID`, `Request_ID`, `Target_ID` and `Recommender_ID` are straightforward. `Categories` is a set of category names that the requestor is interested in enquiring about. `GetPKC` is a Boolean flag which when set to true indicates that the requestor would also like a copy of the target's public key certificate for further communication. If a public-key certificate is available, it is returned in the `Recommendation`, in the `TargetPKC` field.

The `Return_Path` field contains an ordered sequence of recommender IDs. This shows the path through which the Recommendation propagated from the *recommender* to the *requestor*.

The `Recommendation_slip` contains the actual trust information that the requestor is interested in. For each category, there is a sequence containing the `Category_name`, the trust value of the target with respect to this category, and the `Expiry`.

The `Expiry` field contains the expiry date for the `RRQ` or `Recommendation`. In the case of the `RRQ`, this is used to discard any old `RRQ`s that may still be floating around in the system. In the case of each recommendation slip, this is used to indicate the validity period of the recommendation, after which the recommendation should not be relied upon any further.

If the `RRQ` reaches a dead end in its path, and fails to reach a recommender who is able to provide a recommendation, the fields `Recommendation_slip` and `TargetPKC` will be replaced by a `NULL`.

6.2 Protocol flow

The protocol flow is best described using an example, as depicted in Figure 3.

Figure 3 Example: can Alice trust Eric the mechanic?

6.2.1 Requests and Recommendations

Let us assume that Alice (the *requestor*) is requesting a recommendation from Bob (the *recommender*) about Eric (the *target*). Alice is interested in Eric's reputation for servicing cars, especially for VW Golfs, one of which Alice drives (trust category = "Car_Service"). The protocol run is as follows.

1. *Alice->Bob:* *Alice, rrqA01, Eric, [Car_Service], T, 20000101*
2. *Bob->Cathy:* *Bob, rrqB01, Eric, [Car_Service], T, 20000101*
3. *Cathy->Bob:* *Bob, rrqB01, Eric, [Cathy], [(Car_Service,3,20000131)], $(PK_{Eric})_{SK_{Cathy}}$*
4. *Bob->Alice:* *Alice, rrqA01, Eric, [Cathy,Bob], [(Car_Service,3,20000131)], $(PK_{Eric})_{SK_{Cathy}}$*

The protocol is simple and straightforward. Each `RRQ` is sent to the requestor's set of recommenders trusted to recommend in that category in question. In the example above, Alice sends an `RRQ` to Bob because she trusts Bob as a recommender for car servicing mechanics, and Bob trusts Cathy in a similar capacity. Since Bob cannot say anything about Eric with respect to "Car_Service", Bob forwards Alice's `RRQ` to Cathy who may know. Cathy in fact knows about Eric's workmanship, and Cathy believes that Eric's reputation for it is good, i.e. *in Cathy's opinion*, Eric's trust value with respect to category "Car_Service" is 3.

Cathy replies to Bob with a recommendation in message 3. Notice that the Requestor_ID and Request_ID represents the last sender (or forwarder) of the `RRQ` in the forward `RRQ` chain, and not the original issuer of the `RRQ`. This is designed this way to encourage `Recommendations` to be returned using the forward path, which contains at least one trusted node (the original recommender Bob). This also provides, to some degree, anonymity of the original `RRQ` issuer, e.g. as far as Cathy knows, Bob is the requestor of the `RRQ`. Cathy also appends Eric's public key certificate, which is signed by Cathy herself, to the end of the recommendation.

Bob receives the recommendation from Cathy, and changes the Requestor_ID and Request_ID fields. Bob also adds his own ID to the tail of the Return_Path list. He then forwards this to Alice.

6.2.2 Revocation and Refreshing Recommendations

The reputation of entities change over time so there is a need to update the reputation information in the system. The classic method for handling this is through revocation where revocation messages are sent out to revoke certificates. In our trust model there is a need to revoke, as well as *refresh* recommendations. In fact, revoking is a subset of refreshing; they are contained in the same *Refresh* message type. To revoke, a recommender resends the same recommendation with trust value 0. The receiver will treat this as any other 0-value recommendation. Changing the trust value to any other value (1-4) will refresh the recommendation.

In our previous example, if Cathy found out that Eric had made several bad jobs of servicing her car, Cathy may decide that Eric is not trustworthy after all, and would like to inform his previous requestors of this. These messages show how this will be carried out.

5. *Cathy->Bob:* *Eric, [Cathy], [(Car_Service,1,20000131)]*

Bob, upon receiving message 5 also decides to propagate this *Refresh* message to his previous requestors, who, in this example, is just Alice.

6. *Bob->Alice:* *Eric, [Cathy, Bob], [(Car_Service,1,20000131)]*

Figure 4 Refreshing recommendations (arrow points direction of *Refresh* message flow)

Public keys are not included in *Refresh* messages because *Refresh* messages are for refreshing trust, not keys. Keys are just piggybacked on *Recommendations* to avoid another round of protocol for obtaining keys.

The Recommendation Protocol in our trust model makes revocation easier. All that is required is to resend the *Refresh* message to all previous requestors of the same target and category. With traditional certificate mechanisms, the target entity itself carries the certificate, and it is not easy to determine whom it will present the certificate to next, therefore distributing the revocation certificate is harder. Furthermore, since there are potentially more recommenders in our model than CAs in normal certification architectures, there are less agents to broadcast revocations to. This shows how much simpler trust management is through decentralisation.

One major risk in sending *Refresh* messages is the propagation delay of messages through the system. This depends on the availability of the agents in the propagation path and the promptness of each agent in the path at forwarding protocol messages. However, since the protocol is decentralised and each agent may also be a recommender, it is suspected that the availability of the refreshed reputation messages will be higher than in a centralised system.

In short, the *Recommendation protocol makes revocating and refreshing trust information easier and improves availability of Refresh messages*.

7. CALCULATING TRUST

Due to space limitation, in this section, we will show the algorithm used for calculating trust in our model, with a brief description of its use. A more detailed argument for the design of this argument can be found in [AH97].

The trust value of a target for a single given category is computed as follows:

$$tv_p(T) = tv(R1)/4 \times tv(R2)/4 \times .. \times tv(Rn)/4 \times rtv(T) \quad (1)$$

Where,

tv(Ri): Recommender trust value of recommenders in the return path including the first recommender (who received the original RRQ) and the last recommender (who originated the Recommendation).

rtv(T): The recommended trust value of target T given in the recommendation.

$tv_p(T)$: The trust value of target T derived from recommendation received through return path *p*.

tv() must be calculated for each recommendation received on different return paths (1..p), and the final single trust value for target T is the minimum of these calculated trust values:

$$tv(T) = Min(tv_1(T),..,tv_p(T)) \quad (2)$$

We will illustrate this algorithm with our previous example with Eric the mechanic. From the previous example, we have the recommendation path from Cathy to Alice (refer to this as *Rec-path-1*), Cathy → Bob→ Alice, and we have the following trust statement:

- Cathy trusts Eric value 3 (from example)

Assume further that:

- Alice trusts Bob's recommender trust, value 2
- Alice trusts Cathy's recommender trust, value 3

We also assume that Alice had sent out a second RRQ for the same trust category "Car_Service" to David, another of her trusted recommenders, and had received a recommendation from him about Eric (refer to this as *Rec-path-2*).

Alice calculates trust for Eric on *Rec-path-1* as follows:

$$tv_1(Eric) = tv(Bob)/4 \times tv(Cathy)/4 \times rtv(Eric)$$
$$= 2/4 \times 3/4 \times 3$$
$$= 1.125$$

We assume that by using the same algorithm (1), Alice obtains a value of 2.500 ($tv_2(Eric)$ = 2.500) on *Rec-path-2*. Now Alice can apply (2) to obtain the following:

$$tv(Eric) = Min(tv_1(T),tv_2(T))$$
$$= Min(1.125,2.500)$$
$$= 1.125$$

Therefore, from the calculation, Alice may decide that Eric has quite a low trustworthiness as a car-servicing mechanic.

Computing trust is a difficult area, and at this moment the trust computation algorithm above was derived from a great deal of intuition. Nevertheless, a standard algorithm is necessary to lessen ambiguity in trust value recommendations, and so that most *requestors* can be confident that what is received in recommendations comes *close* to a universal[7] standard.

8. DISCUSSION

8.1 Related work

Policymaker [BFL96] is a distributed approach to trust management. Although highly flexible, its implementation is cumbersome for the average user, as the policy definition is a complex procedure. In [YKB93, BBK94], a general formalism was introduced. However, no actual protocol was given, and the calculations are ad hoc. Maurer's [Mau96]

[7] In discussions about areas as subjective as trust, it makes more sense to think of the term *universal* as being constrained by a particular application domain where common standards exist, e.g. the domain of business or finance, instead of taking 'universal' as a synonym for 'global'.

trust model too, although elaborate, is another ad-hoc approach. Marsh gave an elaborate theory of trust in [Mar94]. His techniques are more suited to cooperative autonomous software agents environments.

8.2 Issues ignored

So far, we have ignored a large number of issues in our work, which include provisions for anonymity, entity naming, memory requirements for storing reputations, and the behaviour of the Recommendation Protocol. These issues have been ignored deliberately so that the more complex and understudied area of trust can be satisfactorily pursued, since the issues above are being tackled in work by other researchers. For example, the work in SDSI [RL96] and SPKI [Ell96] includes a novel attempt at eliminating the need for global name spaces.

8.3 Future work

One of our concerns is the lack of understanding of the meaning of trust in computer systems. Currently, we are looking into this problem by surveying the different semantics of trust within areas as diverse as sociology, psychology and philosophy, as well as distributed systems.

There is also a need to look into the need for monitoring and revising trust of other entities, because trust is non-static and non-monotonic.

Finally, we intend to test the behaviour of our protocol and trust calculation algorithms, based on simulations.

9. CONCLUSION

In this paper, we highlighted the need for effective trust management in distributed systems, and proposed a protocol based on *recommendations*. This work, and those being carried out by other researchers, has barely scratched the surface on the issues related to the complex notion of trust. Nevertheless, it is an issue vital to the engineering of future secure distributed systems.

REFERENCES

[AH97] Alfarez Abdul-Rahman, Stephen Hailes. *A Distributed Trust Model*. (To appear) In Proceedings, New Security Paradigms 97 Workshop, September 1997.

[Abd97] Alfarez Abdul-Rahman. *Summary of the DTI's proposed encryption policy*. London School of Economics. http://www.cs.ucl.ac.uk/staff/F.AbdulRahman/docs

[BBK94] Thomas Beth, Malte Borchedring, B. Klein. *Valuation of Trust in Open Networks*. In Proceedings, European Symposium on Research in Computer Security 1994, ESORICS94, pp 3-18.

[BFL96] Matt Blaze, Joan Feigenbaum, Jack Lacy. *Decentralised Trust Management*. In Proceedings, IEEE Conference on Security and Privacy, May 1996.

[DoD85] U.S. Department of Defense. *Department of Defense Trusted Computer System Evaluation Criteria*. DoD 5200.28-STD, 26 December, 1985.

[Ell96] Carl Ellison. *SPKI draft*. http://www.clark.net/pub/cme/spki

[Gam90] D. Gambetta. *Can We Trust Trust?*. In, Trust: Making and Breaking Cooperative Relations, Gambetta, D (ed.). Basil Blackwell. Oxford, 1990, pp. 213-237.

[Jøs96] Audun. Jøsang. *The right type of trust for distributed systems*. In Proceedings, New Security Paradigms 96 Workshop, 1996.

[Luh79] N. Luhmann. *Trust and Power*. Wiley, Chichester, 1979.

[Mar94] Stephen Marsh. *Formalising Trust as a Computational Concept*. Ph.D. Thesis, University of Stirling, 1994.

[Mau96] Ueli Maurer. *Modelling a Public-Key Infrastructure*. In Proceedings, European Symposium on Research in Computer Security 1996, ESORICS96.

[RJ96] Lars Rasmusson, Sverker Jansson. *Simulated Social control for Secure Internet Commerce (position paper)*. In Proceedings, New Security Paradigms '96 Workshop.

[RL96] Ronald Rivest, Butler Lampson. *SDSI – A Simple Distributed Security Infrastructure*.

[RRJ96] Lars Rasmusson, Andreas Rasmusson, Sverker Jansson. *Reactive Security and Social Control*. In Proceedings, 19th National Information Systems Security Conference.

[Sch97] Bruce Schneier. *Why Cryptography Is Harder Than It Looks*. Information Security Bulletin, Vol. 2 No. 2, March 1997, pp. 31-36.

[YKB93] Raphael Yahalom, Birgit Klein, Thomas Beth. *Trust Relationships in Secure Systems - A Distributed Authentication Perspective.* In Proceedings, IEEE Symposium on Research in Security and Privacy, 1993.

[Zur97] Mary Ellen Zurko. *Panels at the 1997 IEEE Symposium on Security and Privacy, Oakland, CA, May 5-7, 1997*. CIPHER, Electronic Issue #22, 12 July, 1997. http://www.itd.nrl.navy.mil/ITD/5540/ieee/cipher/