SPDP: A Secure Service Discovery Protocol for Ad-hoc Networks

Florina Almenárez, Celeste Campo
Dept. Telematic Engineering - University Carlos III of Madrid
Avda. Universidad 30, 28911 Leganés (Madrid), Spain
http://www.it.uc3m.es/pervasive
{florina, celeste}@it.uc3m.es

Abstract— **In ad-hoc networks, mobile devices communicate via wireless links without the aid of any fixed networking infrastructure. These devices must be able to discover services dynamically and share them safely, taking into account ad-hoc networks requirements such as limited processing and communication power, decentralised management, and dynamic network topology, among others. Legacy solutions fail in addressing these requirements.**

In this paper, we propose a new service discovery protocol with security features, the Secure Pervasive Discovery Protocol. SPDP is a fully distributed protocol in which services offered by devices can be discovered by others, without a central server. It is based on an anarchy trust model of PKI and on existing protocols. This way it provides location of trusted services, protection confidential information, secure communications, identification between devices, and service access control.

Keywords— **Ad-hoc networks, service discovery protocol, security, trust.**

I. Introduction

Recent advances in microelectronic and wireless technologies have fostered the proliferation of small devices with limited communication and processing power. They are what are known as "pervasive systems". Personal Digital Assistants (PDAs) and mobile phones are the more "visible" of these kinds of devices, but there are many others that surround us, unobserved. For example, today most household appliances have embedded microprocessors. Each one of these small devices offers a specific service to the user, but thanks to their capacity for communication, in the near future they will be able to collaborate with each other to build up more complex services. In order to achieve this, devices in such "ad-hoc" networks should dynamically discover and share services between them when they are close enough.

In ad-hoc networks composed of limited devices, it is very important to minimise the total number of transmissions, in order to reduce battery consume of the devices. It is also important to implement mechanisms to detect, as soon as possible, both the availability and unavailability of services produced when a device joins or leaves the network. Security in these networks is also critical because there are many chances of misuse both from fraudulent servers and from misbehaving clients.

In this paper, we propose a new service discovery protocol with security features, the Secure Pervasive Discovery Protocol (SPDP). SPDP is a fully distributed protocol in which services offered by devices can be discovered by others, without a central server. It provides location of trusted services, protection confidential information, secure communications, identification between devices, and service access control by forming a reliable ad-hoc network.

The paper is organised as follows: Section II enumerates the main service discovery protocols proposed so far in the literature, we will see that none of them adapts well to ad-hoc networks. Section III presents our secure pervasive discovery protocol, SPDP, with its application scenario, description of the algorithm and security model. Finally, we summarise and list our future research directions in Section IV, including SPDP implementation issues.

II. Related Works

Dynamic service discovery is not a new problem. There are several solutions proposed for fixed networks, with different levels of acceptance, like SLP [1], Jini [2] and Salutation [3]. Recently, other protocols for dynamic environments have appeared: UPnP's SSDP [4] and DEAPspace [5]. Only a few protocols have built-in security, the most important are SSDS [6] and Splendor [7].

However, these solutions can not be directly applied to an ad-hoc network, because they were designed for and are more suitable for (fixed) wired networks. We see three main problems in the solutions enumerated:

- First, many of them use a central server, such as SLP[1], Jini and Salutation. It maintains the directory of services in the network and it is also a reliable entity the security of the system is based on. An ad-hoc network cannot rely upon to have any single device permanently present in order to act as central server, and furthermore, maybe none of the devices present at any moment may be suitable to act as the server.
- Second, the solutions that may work without a central server, like SSDP, are designed without considering the power constraints typical in wireless networks. They make an extensive use of multicast or broadcast transmissions which are almost costless in wired networks but are power hungry in wireless networks.
- Third, security issues are not well covered. SSDS provides security in enterprise environments but may not work in ad-hoc networks with mobile services. Splendor does not provide certificate revocation and trust models of PKIs. They both depend on trustworthy servers and they propose solutions which are provided at the IP level.

Accepting that alternatives to the centralised approach are required, we consider two alternative approaches to distributing service announcements:

- The "Push" solution, in which a device that offers a service sends unsolicited advertisements, and the other devices listen to these advertisements selecting those services they are interested in.
- The "Pull" solution, in which a device requests a service when it needs it, and devices that offer that service answer the request, perhaps with third devices taking note of the reply for future use.

In ad-hoc networks, it is very important to minimise the total number of transmissions, in order to reduce battery consume. It is also important to implement mechanisms to detect as soon as possible both the availability and unavailability of services produced when a device joins or leaves the network. These factors must be taken into account when selecting between a push solution or a pull solution.

The DEAPspace algorithm is the only service discovery protocol, listed above, that tries to minimise the total number of transmissions. It uses a pure "push" solution and each device periodically broadcast its "world view" although none of them has to request a service.

In this paper we propose a new service discovery algorithm, the Secure Pervasive Discovery Protocol (SPDP), which merges characteristics of both pull and push solutions. This way we think a better performance can be achieved. On the other hand, SPDP provides security based on existing protocols and an anarchy trust model of PKI. An anarchy trust model does not require neither a central trusted server nor hierarchy architecture being suitable to overcome the challenges imposed by ad-hoc networks such as no central management, no strict security policies and highly dynamic nature.

III. SPDP: Secure Pervasive Discovery Protocol

The Secure Pervasive Discovery Protocol (SPDP) is intended to solve the problem of enumerating the services available in single hop ad hoc networks, composed of devices with limited transmission power, memory, processing power, etc. Legacy service discovery protocols use a centralised server that listens for broadcast or multicast announcements of available services at a known port address, and lists the relevant services in response to enquiries. The protocol we propose does away with the need for the central server. Ad-hoc networks cannot rely upon to have any single device permanently present in order to act as central server, and further, none of the devices present at any moment may be suitable to act as the server.

[1] SLP supports a distributed mode, but usually the implementations use centralised mode

One of the key objectives of the SPDP is to minimise battery use in all devices. This means that the number of transmissions necessary to discover services should be reduced as much as possible. A device announces its services only when other devices request the service. Service announcements are broadcasted to all the devices in the network, all of which will get to know about the new service simultaneously at that moment, without having to actively query for it.

In addition, SPDP enables to share services safely, through a trust model between devices which act like its own Certification Authority (CA). Both the servers and the clients are protected from malicious devices.

In the remainder of this section, we present the application scenario for SPDP and some considerations to be taken into account. Then, we will formally describe the algorithm used to implement it, and we will discuss the security support of SPDP based in a new trust model for open distributed environments.

A. Application scenario

Let's assume that there is an ad-hoc network, composed of `D` devices, each device offers `S` services, and expects to remain available in this network for `T` seconds. This time `T` is previously configured in the device, depending on its mobility characteristics.

Each device has an SPDP User Agent (SPDP_UA) and an SPDP Service Agent (SPDP_SA). The SPDP_UA is a process working on the user's behalf to search information about services offered in the network. The Service Agent SPDP (SPDP_SA) is a process working to advertise services offered by the device. The SPDP_SA always includes the availability time `T` of its device in its announcements.

Each device has a cache associated which contains a list of the services that have been heard from the network. Each element `e` of the cache associated to the SPDP_UA has three fields: the service description, the service lifetime and the service expiration time. The service expiration time is the time it is estimated the service will remain available. This time is calculated as the minimum of two values: the time the device has promised to remain available, `T`, and the time the server announced that the service would remain available. Entries remove themselves from the cache when their timeout elapses.

With regard to security, each device handles a list of reliable devices and the trust degree associated with them. Depending on the trust degree, a device decides to store the service offered by a device on its cache. When the devices go to access to the service, it select first the device with biggest trust degree, also.

B. Algorithm description

The SPDP has two mandatory messages: `SPDP Service Request`, which is used to send service announcements and `SPDP Service Reply`, which is used to answer a SPDP Service Request, announcing available services. SPDP has one optional message: `SPDP Service Deregister`, which is used to inform that a service is no available longer.

Now, we will explain in detail how SPDP_UA and SPDP_SA use these primitives.

B.1 SPDP User Agent

When an application or the final user of the device needs a service, whether a specific service or any service offered by the environment, it requests the service from its SPDP_UA (Fig. 1).

If a specific type of service has been requested:

- The SPDP_UA searches for that service in the list of local services and in its cache. If it is found, the SPDP_UA gives the application the service description.
- If it is not found, the SPDP_UA broadcasts a SPDP Service Request for that service, and it waits `CONFIG_WAIT_RPLY` seconds for replies. If no reply arrives, the SPDP_UA answers to the application that the service is not available in the network. If some reply arrives, the SPDP_UA updates its cache accordingly. In order to minimise the spreading of service announcements of malicious devices, the SPDP_UA does not store services offered by untrustworthy devices. Finally, it gives the application the service descriptions of trusted servers received.

In order to allow an application to request its SPDP_UA to discover all available services in the network, we introduce a new type of service: service `ALL`. Whenever an application requests for all

```
search(s) {
  foreach (l $\in$ Local)
    if (l.type == s.type) result +=l;
  foreach (e $\in$ Cache)
    if (e.type == s.type) result += e;
  if (result.length != 0) return(result);
  broadcast(SPDPServiceRequest(s.type));
  timeout tout = CONFIG_WAIT_RPLY;
  loop (forever) {
    list_remote = hear_network(SPDPServiceReply$_s$);
    if (expired tout) {
      break;
    } else {
       update_cache(list_remote);
       result += list_remote;
    }
  }
  return(result);
}


search(ALL) {
  broadcast(SPDPServiceRequest(ALL));
  timeout tout = CONFIG_WAIT_RPLY;
  loop (forever) {
    list_remote += hear_network(SPDPServiceReply$_{ALL}$);
    if (expired tout)
      if (list_remote.length == 0) delete_cache();
      break;
    else
       update_cache(list_remote);
  }
  result = Cache + Local;
  return(result);
}
```

```
update_cache(list) {
  foreach (l $\in$ list)
    if (trust_degree(l.IP) $\geq$ 0.5)
       Cache += l;
}

trust_degree(ip) {
  if $\exists$ trust degree of l.IP return this
  else
    goto trust_formation
}
```

Fig. 1. SPDP_UA pseudocode implementation

available services in the network, the SPDP_UA sends a SPDP Service Request searching a service type `ALL`, and waits `CONFIG_WAIT_RPLY` seconds for the reply:

- If a reply arrives, the SPDP_UA updates the cache accordingly and answers to the application listing the local services plus the services in the cache.
- If no reply arrives, the SPDP_UA deletes all services stored in the cache and replies to the application listing only the local services.

The SPDP_UA in all devices are continually listening on the network for all types of messages (SPDP Service Requests and SPDP Service Replies). Whenever a SPDP Service Reply announcing a service is received, the SPDP_UA update its cache accordingly (Fig. 1).

Moreover, the device's cache has a limited size. When an SPDP_UA hears a new announcement but the cache is full, it deletes the service entry offered by the device with less trust degree or less expiration time. In the next section we will explain how SPDP_UA calculates the trust degree of the devices.

B.2 SPDP Service Agent

The SPDP_SA advertises services offered by the device. It has to process SPDP Service Request messages and to generate the corresponding SPDP Service Reply, if necessary (Fig. 2).

```
receive(SPDPServiceRequest(s.type)) {
  foreach (l $\in$ Local)
    if (l.type == s.type) result += l;
  foreach (e $\in$ Cache)
    if (e.type == s.type) result += e;
  if (result.length != 0) {
    timeout tout = generate_random_time($\frac{1}{T}$);
    loop (forever) {
      list_remote += hear_network(SPDPServiceReply$_s$);
      if (expired tout) {
        update_cache(list_remote);
        if not (result $\subseteq$ list_remote) {
          result = result - (list_remote $\cap$ result);
          broadcast(SPDPServiceReply(result));
          exit;
        }
      }
    }
  }
}


receive(SPDPServiceRequest(ALL)) {
  timeout tout = generate_random_time($\frac{1}{T*Cache.size}$);
  result = Local + Cache;
  loop (forever) {
    list_remote += hear_network(SPDPServiceReply$_{ALL}$);
    if (expired tout) {
      update_cache(list_remote);
      if not (result $\subseteq$ list_remote) {
        result = result - (list_remote $\cap$ result);
        broadcast(SPDPServiceReply(result));
        exit;
      }
    }
  }
}
```

Fig. 2. SPDP_SA pseudocode implementation

When a SPDP_SA receives a SPDP Service Request with a service type different of service `ALL`:

- First, it checks whether the requested service, S, is one of its local services, or it is in the cache.

- If it is, it generates a random time t, inversely proportional to the availability time of the device, T. So, the more time the device is able to offer the service, the higher the probability of the device answering first.
- During this time, the SPDP_SA listens the network for any SPDP Service Reply of the same request and it updates the cache accordingly.
- When the timer expires, if the SPDP_SA knows about some additional devices offering the service `S` and that have not been announced yet, it sends its SPDP Service Reply.

When a SPDP request for all services (service type `ALL`) is received, then the SPDP_SA:

- Generates a random time t, inversely proportional to the availability time of the device, T, and to the number of elements stored in the cache of that interface. So, the more time the device is able to offer the service and the bigger the cache, the higher the probability of answering first. We suppose the device with the highest availability time and the bigger cache is the one with the most accurate view of the world.
- During the interval t, the SPDP_SA listens to the network for any SPDP Service Reply of the same request and it updates the cache accordingly.
- When the timer expires, if the SPDP_SA knows about some new services that have not been announced yet, it sends its SPDP Service Reply, listing the local services plus the services in the cache.

In certain network technologies, it is possible to detect when a device is switched off or it roams to other network. If this happens, its SPDP_SA has to send a SPDP Service Deregister, listing all its local services. When a SPDP_UA hears this message, it deletes these services of its cache.

C. Security Issues

SPDP provides authentication, authorisation, data integrity, confidentiality, and non-repudiation through an anarchy PKI.

Mutual authentication is based on a challenge mechanism between devices. When a foreign device wants to join to the ad-hoc network, we must look for any trusted device that entrusts it, in order to establish a distributed trust. If no one knows it, then we must decide whether rely on it or not. From authentication, we use digital signatures for non-repudiation purpose.

Authorisation to the services is granted according to security policies. This policies determine the trust degree that a client must be for access to the offered service.

Data integrity and confidentiality are guaranteed by IPSec [8]. SPDP Service Request and SPDP Service Reply messages are protected in order to guarantee their integrity and confidentiality.

We propose an anarchy trust model in which every device acts as its own CA, issuing certificates for its own services; therefore, it is a decentralised trust model. Trust relationships are established between CAs, this way we ensure the storage of reliable service information in our cache. That is, a device stores only in its cache the service information originated from other trusted user agents and service agents.

Each device handles a list of reliable devices and the trust degree associated with them. This model is very simple indeed since, unlike in SSDS and Splendor, the trust degree are established only between two components.

We choose an anarchy trust model due to its advantages. The more important ones are: a new device can be easily incorporated, when a device is compromised the security of the hole network is not, and multiple trust paths between devices can be defined. All this avoids the dependence on a central server.

C.1 The Trust Model

In the last decades, some trust models, based on hierarchy PKI, have been defined such as [9], [10], [11], [12]. These models do not consider that the entities interacting are autonomous and mobile. They do not define a dynamic trust model along time and always depend upon a root server. These characteristics are not suitable for ad-hoc networks. But, what does trust mean?

Different definitions of trust are given in psychology, sociology, economics and mathematics. Based on the definition of trust given by Josang [13], we define trust as: "A belief that one entity[2] has about another entity; there must be a reason behind the belief such as past experiences, knowledge about entity's nature or recommendations. The belief represents how an entity will behave or

[2] In this paper entity is equivalent to device

perform an specific task which implies a potential hazard".

Thus, trust can be expressed as a belief about another entity in terms of its rigth behaviour. It is a subjective concept, being a personal opinion based primarily on first observations or carefully considered advice from others if available, allowing decisions to be made with only partial knowledge.

To formalise a decentralised trust model we have identified three properties of the trust relationships. Let A, B and C be the devices in a ad-hoc network. We define the trust relationship function between two devices A and B, R(A,B). R is a continuous function with values between 0 and 1. We use fuzzy logic rather than the usual boolean logic. 0 and 1 are extreme cases, but values in between are also possible, for example, 0 for distrust, 1/2 for ignorance and 1 for trust. The properties of R are the following:

Reflexive Every device trust on itself ($R(A, A) = 1$).

Non-symmetrical If A trusts on B, not necessarily B trusts on A. If we have $R(A, B) = \beta \wedge R(B, A) = \gamma \nRightarrow \beta = \gamma$.

Conditionally transitive If A trusts on B and B trusts on C, then A conditionally trusts on C. In mathematic terms: it exists the pairs $R(A, B) = \beta \wedge R(B, C) = \gamma \Rightarrow R(A, C) \leq \beta.\gamma$.

The trust model architecture (Fig. 3) has five modules: context, trust formation, trust evolution, communication and system trust and trust relationships. In accordance with the trust definition, given an specific context, first a device forms an initial trust value. Next, this value may change (increase or decrease) depending on the device behaviour.

C.1.a Trust Formation. Initially new devices have no evidence of past experiences to establish an initial trust value for interaction. There are two sources to form an initial trust degree: personal opinion (direct trust) or recommendations (indirect trust).

Direct Trust . The trust value is formed from an initial threshold, for example, ignorance value 0.5. This value could be increased either by getting some information about the device's nature i.e. device type, owner, etc., or by human intervention.

Indirect Trust . It is applied when there exists trust relationships between some devices. The trust value is created from recommendations which are given by third trusted parties. We consider third trusted parties whose trust value is bigger than a threshold α, where $\alpha \geq 0.5$.

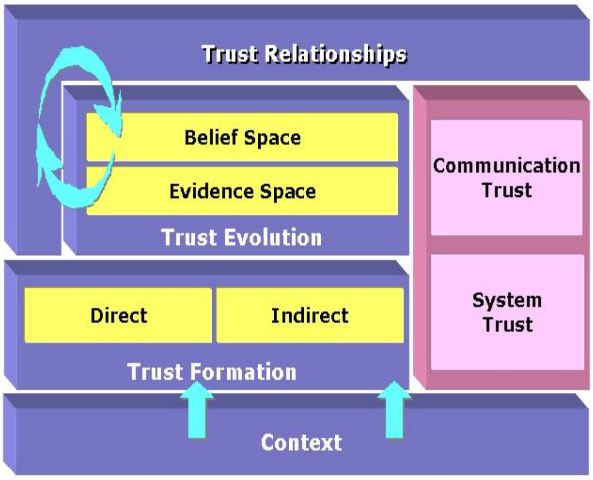

Fig. 3. Trust Model Architecture

We assume that certificates issued by third trusted parties are a recommendation mechanism, too. In this case, recommendation value $R(D_i)$ would be α and otherwise the recommendation would be the trust degree sent by third trusted party.

We calculate the trust value of device B (T_B) by making a weighted average:

$$T_B = \frac{1}{\sum_{i=1}^{n} T_i} \sum_{i=1}^{n} R(D_i) * T_i$$

where $R(D_i)$ is recommendation value from device i, T_i is its trust value, and n is the recommender number.

Trust information is shared between trustworthy devices through a recommendation protocol. The recommendation protocol has three messages: SPDP Recommendation Request, SPDP Recommendation Reply and SPDP Recommendation Alert. SPDP Recommendation Request is used to request recommendations about other devices; with SPDP Recommendation Reply we answer to this request. When we are attacked by a device, then we use SPDP Recommendation Alert in order to warn all devices within the network about it. For example, when a device sends us multiple SPDP Service Request during a very short period of time. In this case, the device is considered unreliable.

Fig. 4 shows the algorithm to request and reply recommendations. When we detect an unknown device (target) we broadcast a recommendations request from other devices. We wait a time x for the replies. When recommendations are received, we recalculate the trust value taking into account only the replies from trusted devices (recommenders).

```
trust_recommendation(Target_ID) {
  broadcast(SPDPRecommendationRequest(Target_ID));
  timeout x;
  loop (forever) {
    R = hear_network(SPDPRecommendationReply$_{Target\_ID}$);
    if T$_{recommender}$ $\geq \alpha$ {
      Trust += R*T$_{recommender}$;
      Total$_{rec}$ += T$_{recommender}$;
    }
    if (expired x) exit;
  }
  if (Total$_{rec}$ == 0) Trust = 0.5;
  else Trust = $\frac{1}{Total_{rec}}$*Trust;
  return Trust;
}
```

Fig. 4. Trust Recommendation Protocol pseudocode implementation

C.1.b Trust Evolution. As we mentioned previously, trust learning is gradual and dynamic, since the trust degree changes along time. In fact, it is often a consequence of a complex set of beliefs, perceptions and interpretations. Trust value changes according to positive and negative experiences in an specific context.

Therefore, we calculate a new trust value T_{i+1} taking into account both past and present, that is, the previous trust value T_i and the value of the actions V_a that represents device's behaviour. We also define a parameter β as the weight of the past, with $0 < \beta < 1$.

$$T_{i+1} = \beta * T_i + (1 - \beta) * V_a$$

With this function, the trust value increases or decreases according to the device's behaviour.

Positive and negative actions are considered evidences. Thus, we build our evidence space that are facts in the knowledge base of each device. These facts are useful to identify trustworthy and untrustworthy devices. It is important to identify untrustworthy devices because mistrust is different of a simple absence of trust (ignorance).

As a consequence of the evidence space, we create our belief space. This space represents beliefs about a device which allow us to establish rules of access control to the services offered by the devices.

This way, we establish a reliable ad-hoc network within which we may use services and exchange information without risk of deceit.

IV. Conclusions and Future Work

Ad-hoc networks are becoming increasingly common thanks to the development of mobile device technology. When a device connects to an ad-hoc network, it wants to know the services offered by the network and in turn it may offer its own services. Client applications in the device want to discover trustworthy services automatically, while server applications want to be used by trustworthy clients that will not misuse or attack them. Additionally, secure network communication is also an important issue. These goals are carried out by SPDP.

SPDP is a good service discovery protocol for ad-hoc networks since:

- It is based on a distributed open architecture, therefore, it does not require central servers;
- It has a simple architecture which contains only two type of components, user agents and service agents;
- It provides autonomous and mobile agents with an simple method for discovering services that are available;
- It minimises battery use in all devices since the number of transmissions necessary to discover services is reduced as much as possible;
- It integrates a security model in order to guarantee the required security level by devices. Security issues includes mutual authentication between devices, data integrity, confidentiality, non-repudiation, and access control based on degree of trust.

Thus, we fulfil the challenges imposed by ad-hoc networks.

In order to evaluate the SPDP performance in different mobility scenarios, we have implemented the SPDP protocol on the well-known simulator Network Simulator [14]. Now, we are obtaining the first simulation results, and we are carrying our experiments:

- To obtain the number of messages transmitted to discover an available service in the ad-hoc network, and to compare that with other protocols,
- To determine the probability of discovering a service that is not available in the ad-hoc network.
- To evaluate security issues on different scenarios.

Besides that, we are also implementing the SPDP for Java 2 Micro Edition (J2ME) in order to deploy it in real devices.

Finally, we are studying IPSec for ad-hoc networks and its interoperability with our trust model.

Acknowledgments

The authors thank their contributions to Carlos García-Rubio and Andrés Marín.

References

[1] "Service location protocol, version 2", 1999.

[2] "Jini Architectural Overview. White Paper", 1999.

[3] Brent A. Miller and Robert A. Pascoe, "Salutation service discovery in pervasive computing environments", Tech. Rep., IBM White Paper, Feb. 2000.

[4] Yaron Y. Goland, Ting Cai, Paul Leach, and Ye Gu, "Simple service discovery protocol/1.0", Tech. Rep., Microsoft, 1999.

[5] Michael Nidd, "Service Discovery in DEAPspace", *IEEE Personal Communications*, Aug. 2001.

[6] Steven E. Czerwinski, Ben Y. Zhao, Todd D. Hodes, Anthony D. Joseph, and Randy H. Katz, "An architecture for a secure service discovery service", in *Proc. Mobicom'99*, 1999.

[7] F. Zhu, M. Mutka, and L. Ni, "Splendor: A secure, private, and location-aware service discovery protocol supporting mobile services", in *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications (Percom'03)*. IEEE Computer Society, Mar. 2003, pp. 235–242.

[8] S. Kent and R. Atkinson, "Security architecture for the internet protocol (IPSec)", NOV 1998.

[9] T. Beth, M. Borcherding, and B. Klein, "Valuation of trust in open networks", in *Proceedings of the European Symposium on Research in Computer Security (ESORICS '94, Brighton, UK)*, Heidelberg, Germany, Nov. 1994, number 875 in Lecture Notes in Computer Science, pp. 3–18, Springer-Verlag.

[10] Matt Blaze, Joan Feigenbaum, and Jack Lacy, "Decentralized trust management", in *Proceedings of the IEEE Symposium on Research in Security and Privacy*, Oakland, CA, May 1996, IEEE Computer Society, Technical Committee on Security and Privacy, number 96-17, IEEE Computer Society Press.

[11] Alfarez Abdul-Rahman and Stephen Hailes, "A distributed trust model", in *Proceedings of the ACM Workshop on New Security Paradigms*, Cumbria, United Kingdom, Sept. 1997, pp. 48–60, ACM SIGSAC, ACM Press.

[12] A. Jøsang and S. J. Knapskog, "A metric for trusted systems", in *Proc. 21st NIST-NCSC National Information Systems Security Conference*, 1998, pp. 16–29.

[13] A. Jøsang, "An algebra for assessing trust in certification chains", in *Proceedings of the Network and Distributed Systems Security (NDSS'99) Symposium, The Internet Society*, 1999.

[14] "The network simulator - ns-2".