# Secure Long Term Communities In Ad Hoc Networks

Nicolas Prigent
Thomson R&I / Supélec
1, Avenue de Belle Fontaine - CS 17616
35576 Cesson-Sévigné
France
nicolas.prigent@thomson.net

Christophe Bidan
Supélec
BP 81127
35511 Cesson-Sévigné
France
christophe.bidan@supelec.fr

Jean-Pierre Andreaux
Thomson R&I
1, Avenue de Belle Fontaine - CS 17616
35576 Cesson-Sévigné
France
jean-pierre.andreaux@thomson.net

Olivier Heen
Thomson R&I
1, Avenue de Belle Fontaine - CS 17616
35576 Cesson-Sévigné
France
olivier.heen@thomson.net

## ABSTRACT

Until recently, ad hoc networks were mainly used for military and security-sensitive applications. Nowadays, they could also be used in SOHO (Small Office / Home Office) or home networks. In such networks, devices are linked by long term relations. To ensure their security, it is necessary to define precisely which devices belong to a given network and are consequently inside the security perimeter. The chosen mechanisms need to be easy to use, because the users of SOHO and home networks are neither willing nor able to configure the security of their network. In this paper, we present a new fully distributed approach for securing long term communities of devices in SOHO and home ad hoc networks that minimizes user intervention.

**Keywords:** ad hoc networks security, secure long term community, home network security.

## 1. INTRODUCTION

Advances have been made recently in networking and computing in such a way that anybody can benefit from these technologies without being an expert. Ad hoc networks [7] are symptomatic of this fact. They consist in groups of devices that offer network connectivity without requiring any infrastructure or extensive configuration.

By opposition to infrastructure-based networks, ad hoc networks have properties [1, 9] that impact on the use of classical security solutions. First, their topology can change at any time. There is no guarantee of device connectivity, and no device can be assumed to be always present. Consequently, there can be no centralized point of control. Moreover, the boundaries of ad hoc networks are poorly defined, by opposition to the one of wired LANs that can be enforced using the infrastructure.

Until now, research in ad hoc networks security mainly focused on secure routing protocols [10, 23] and secure transient relations between devices [3, 6, 9, 21, 22]. These topics are particularly adapted to the military and security-sensitive fields, that were the major domains of application of ad hoc networks.

However, ad hoc networks could greatly benefit to SOHO (Small Office / Home Office) networks and home networks [4, 9], that do not only focus on transient relations. In SOHO and home ad hoc networks[1], communicating devices (e.g. computers, TV sets, set-top boxes, PDAs, printers, etc.) are linked by long term relations. While, like in any classical ad hoc network, the topology could evolve quite frequently, and even if some devices could be temporarily unreachable, the devices composing the network will not vary very often: a device enters the network for an *a priori* long time, e.g. when a householder buys a new device and adds it into the home network, and leaves it *a priori* definitively, e.g. when this device is sold, lost or stolen. In order to secure SOHO and home networks, the first step is to define which devices belong to the network, i.e. which devices are inside the security perimeter and are authorized to access the services offered to the members of the network. Once these devices are identified and authenticated, it is possible to set up secure relations between them.

Contrary to military and security-sensitive fields, that can benefit from extensive administration skills whenever necessary, resources and skills in security administration are scarce in SOHO and home networks: users neither have time nor skills to manage security. Although being the network authority, they are not experts. Worse, they are often considered as the weakest link of security [20].


Proceedings of the f[t] ACM Workshop Security of Ad Hoc and Sensor Networks Fairfax, Virginia


[1]Starting from here, we will use "SOHO and home networks" for short, implying "SOHO and home ad hoc networks".

In this paper, we introduce the notion of secure long term community as a way to mark the boundary between the devices belonging to a given network and the others, taking into consideration the different aspects of ad hoc networks and their requirements for ease of use in home and SOHO environments. More particularly, we propose a mean to set up and manage a long term virtual private network between devices placed under a common policy.

In section 2, we define the concept of secure communities as well as the required security properties they should offer. In section 3, we describe some current existing models and their limitations regarding these required security properties. Our proposal is given in section 4. As a conclusion, we summarize its advantages in section 5.

# 2. SECURE LONG TERM COMMUNITIES

Albers et al. [1] define a community as "a set of devices related by a trust relation". In SOHO and home networks, the devices belonging to the same authority (e.g., the communicating devices in the same household) can reasonably trust each others, and consequently form a community. Because the trust relation between the devices lasts for an *a priori* long time, this is a long term community.

All the elements of a SOHO or home network do not necessarily strictly belong to one and the same person. However, the different owners of the devices share the same interest in connecting them altogether. For example, in a home network, each family member can have its own devices on which she or he makes authority. However, the global interest of having the devices in the household sharing data and services is common to the whole family. In this paper, and according to [14] that states a community as "a group linked by a common policy", we consider the case of possibly multiple authorities sharing the same policy.

Thus, we define a secure long term community as a set of devices:

- that share a trust relation,
- that have been authorized by their authorities to communicate with each others for an *a priori* long time,
- that are able to do so securely.

In this paper, and by opposition to [6], we do not consider that a device corresponds to a given user and *vice versa*. A user can own multiple devices (e.g. a PDA, a mobile phone and a TV set), and multiple users can share authority on a single device, like a TV set.

## 2.1 Security services offered to the members of the community

From a security point of view, a secure long term community is a virtual subnetwork involving the devices of the community and able to protect itself against eavesdropping, modification and injection of messages [1]. Thus, a secure community offers merely the same services as a (Virtual) Private Network:

- Authentication of devices: a device is able to check if another one belongs to its community.
- Confidentiality of communications between members: an attacker must not be able to eavesdrop messages.
- Authenticity of communications between members: an attacker must not be able to modify, to inject or to replay messages.

We will not consider the requirement for secure routing in this paper. In our case, routing is a service proposed by the underlying network technology. If all communications are efficiently encrypted and authenticated, the only attack that can still occur is a denial of service.

Moreover, extensive research [5, 10, 23] has already addressed secure routing problems.

## 2.2 Robustness regarding physical constraints

Like in classical ad hoc networks, devices connectivity cannot be guaranteed in SOHO and home networks. Two devices belonging to the same community may not be able to communicate at a given time. To establish this, we introduce the following notations:

- Let $\Lambda$ be a community and $a$ a device, $a \in \Lambda$. We call $\Lambda(a)$ the devices belonging to $\Lambda$ that $a$ can authenticate as so, i.e. the knowledge $a$ has of $\Lambda$.
- We call $\Phi(a)$ the devices that $a$ can communicate bidirectionally with at a given time. N.B.: All the devices of $\Phi(a)$ do not necessarily belong to $\Lambda$, $a \in \Lambda$.
- We call $\Delta(a)$ the devices belonging simultaneously to $\Lambda(a)$ and $\Phi(a)$: $\Delta(a) = \Phi(a) \cap \Lambda(a)$.

Because all the devices of the community may not be reachable at one time, we introduce the following requirements:

REQUIREMENT 1. *Any two devices 'a' and 'b' of the same community $\Lambda$, $a \in \Lambda(b)$ and $b \in \Lambda(a)$, that can communicate should be able to communicate securely, even when no other device of the community is reachable.*

For instance, mobile devices such as a PDA and a mobile phone should be able to communicate securely even when the user is away from home and no other device is available.

REQUIREMENT 2. *A community should be able to evolve without requiring a specific member.*

Particularly, any two devices should be able to get into the same community while not able to reach any other device from their own community[2]: a user having only his or her PDA while not at home, and buying a mobile phone should be able to insert it in the PDA's community, even if no other device is available.

Moreover, it should be possible to remove any device from the community without this community to collapse.

## 2.3 Secure evolution of the community

A community will evolve along the insertion and removal of devices. This evolution must be secure to maintain the security of the community. In this sense, the secure evolution of the community is another security service it offers. A community presents four possible evolution operations: initialization, insertion, removal and banishment.

REQUIREMENT 3. *A community should be able to evolve securely according to the evolution operations described thereafter.*

[2]This requirement makes trivially unsuited the proposals (such as [19, 23]) where more than one device is required to make the community evolve.

#### 2.3.1 Initialization

A community has an initial state (e.g. when the first device is bought), from which it can evolve. The security of the community should be ensured in this initial state, and should be maintained while the community evolves. Let $\Lambda$ be the community, and $a$ the first element of this community. The initialization is defined by:

$$\textbf{Init: } \Lambda := \{a\}$$

#### 2.3.2 Insertion and merge

A community can accept new members. After a new member enters the community, it is able to identify the other members of this same community as belonging to it, and the other members also identify it as a member of the community. After the insertion of $b$ in a community $\Lambda$, we have:

$$\textbf{Insert: } \Lambda := \Lambda \cup \{b\}$$

However, all the devices may not be reachable during the insertion. Consequently, the unreachable devices may not be aware of the modification of the community. Despite this fact, for all $a$ in $\Lambda$ reachable by $b$ during $b$'s insertion, we should guarantee that:

$$\textbf{Insert: } \forall a \in \Phi(b) \cap \Lambda,\ \Lambda(a) := \Lambda(a) \cup \{b\}$$
$$\text{and } \Lambda(b) := \Lambda(b) \cup \Lambda(a)$$

Thus, after the insertion, $b$ knows all the devices of the community locally reachable, as well as all the devices they themselves know as belonging to the community.

The insertion of a device in a community can be generalized to the merge of two communities: let $a$ be a device belonging to the community $\Lambda$, and $b$ another device belonging to the community $\Lambda'$. The community resulting from the merge of the two communities $\Lambda$ and $\Lambda'$ contains the devices of both these communities.

$$\textbf{Merge: } \Lambda := \Lambda' := \Lambda \cup \Lambda'$$

Some devices belonging to $\Lambda$ or $\Lambda'$ may be unreachable during the merge of the two communities. Consequently, we should simply guarantee that all reachable devices in $\Lambda$ and $\Lambda'$ have the same knowledge of the community. If $a \in \Lambda$ and $b \in \Lambda'$ are two devices reachable during the merge, we have:

$$\textbf{Merge: } \forall c \in \Delta(a), \forall d \in \Delta(b),\ \Lambda(c) := \Lambda(c) \cup \Lambda'(d)$$
$$\text{and } \Lambda'(d) := \Lambda'(d) \cup \Lambda(c)$$

#### 2.3.3 Removal

A device can also leave its community, for example when a user sells it. During the removal operation, the user can access the device, because the device to be removed is available. After the removal, this device should not be recognized as a member of the community anymore. Similarly, the removed device should consider itself as removed. After the removal of $b$ from the community $\Lambda$, we have:

$$\textbf{Remove: } \Lambda := \Lambda \setminus \{b\}$$
$$\text{and } \Lambda' := \{b\}$$

where $\Lambda'$ is a distinct community containing only $b$.

However, all the devices may not be reachable during the removal, and the unreachable devices may not be warned of the modification of the community. Despite this fact, for all $a$ reachable by $b$ in $\Lambda$ during $b$'s removal, we should guarantee that:

$$\textbf{Remove: } \forall a \in \Delta(b), \Lambda(a) := \Lambda(a) \setminus \{b\},$$
$$\text{and } \Lambda'(b) := \{b\}$$

We consider that if more than one device have to be removed from the community, they will be removed separately.

#### 2.3.4 Banishment

When a device is lost or stolen, it has to be removed from the community, otherwise a malicious person who owns it will use it to access the rest of the community. Banishment consists in removing a member from a community without being able to access it: by opposition to the removal operation, the device to be banished is not available. After the banishment of $b$ from the community $\Lambda$, we have:

$$\textbf{Banish: } \Lambda := \Lambda \setminus \{b\}$$

However, all the devices may not be reachable during the banishment. Consequently, the unreachable devices may not be aware of the modification of the community. Despite this fact, $c$ being a device of the community reachable by the user during $b$'s banishment, for all $a$ in $\Delta(c)$, we should guarantee that:

$$\textbf{Banish: } \forall a \in \Delta(c), \Lambda(a) := \Lambda(a) \setminus \{b\}$$

Unlikely to the removal operation, we cannot act on $b$ for banishment.

We consider that if more than one device have to be banished from the community, they will be banished separately.

### 2.4 Global and loose consistency of communities

If at any time, each device in the community had the same complete knowledge of the community, we would reach the global consistency of the community:

$$\textbf{Global consistency: } \forall a \in \Lambda, \Lambda(a) = \Lambda$$

Because some devices may be unreachable during the evolution of the community, it may not be possible to ensure global consistency. Loose consistency consists in the fact that, $a$ being member of the community $\Lambda$, all the devices of $\Delta(a)$ share the same knowledge $\Lambda(a)$, and $\Lambda(a)$ is the most up-to-date knowledge of $\Lambda$ in $\Delta(a)$ . The loose consistency algorithm should locally ensure that:

$$\textbf{Loose consistency: } \forall a \in \Lambda, \forall b \in \Delta(a), \Lambda(a) = \Lambda(b)$$

If all the devices of a community communicate altogether again, we reach global consistency of the community.

REQUIREMENT 4. *A community may split physically in an arbitrary number of partitions (possibly as many as there are devices), each of them evolving independently. When two partitions can communicate again, their knowledge of the community must get consistent.*

Using the example of Requirement 2, when the user will be back home, his or her other devices will have to take into account the fact that the mobile phone has been inserted.

## 3. THE CURRENT MODELS

In ad hoc networks, the infrastructure cannot be used to ensure security. Thus, we have to build a Virtual Private Network between the members of the community. A VPN enables its members to communicate securely over an uncontrolled network (e.g. the Internet or a wireless medium) through the use of cryptography. The problem is to define the relevant cryptographic materials to be set in the elements of a community, and the way this material should be managed.

Four traditional models are currently used that offer the required security services to the members of a community. In this section, we inquire if they fulfill Requirements 1 to 4, and consequently if they are suitable for establishing a secure long term community in SOHO and home networks.

### 3.1 Preshared common data model

### 3.1.1 Scheme

In this model, community members share the same data (e.g. a secret key), that "represents" the community. A device belonging to the community knows this data, and a device knowing this data belongs to the community. Let $\Lambda$ be the community and $k$ the preshared common data, we have:

$$\Lambda = \{a \mid a \; knows \; k\}$$

where "$a \; knows \; k$" means that the device $a$ is able to prove that it knows $k$. The global consistency of the community (Requirement 4) is ensured since we have:

$$\forall a \in \Lambda, \Lambda(a) = \{b \mid b \; knows \; k\} = \Lambda$$

that is, a device $a$ trusts any device $b$ able to prove it knows $k$.

### 3.1.2 Examples

The preshared common data model is used in 802.11 [12] wireless networks technology, that is often used in ad hoc networks: the WEP key is used to ensure authenticity and confidentiality of messages, as well as to authenticate devices trying to access the network when the "shared key authentication" is activated. Devices knowing the WEP key belong to the network.

### 3.1.3 Limitations

The members of a community sharing information known only by the community members, they can authenticate each other and ensure messages confidentiality and authenticity.

The initialization of the community is made when the shared information $k$ is chosen (possibly randomly) and inserted in the first device of the community. A user sets $k$ in a device to insert it in the community, and erases $k$ from a device to remove it.

In order to ban a device, a user would have to erase $k$ from it. By assumption, she or he cannot access it for banishment. The only solution for the banishment of the device is to change $k$ in each device not banished, thus in fact creating a new community, which becomes clearly impractical when the devices get numerous. Consequently, the preshared common data model does not enable the banishment of a device, and thus does not fulfill Requirement 3.

## 3.2 Preshared derived data model

### 3.2.1 Scheme

In the preshared derived data model, a seed information *seed* represents the community. Each community member can be uniquely identified because *seed* is derived for each of them, by opposition to the preshared common data model. Each community member has the required information to prove it belongs to this community and to check any entity claiming to do so.

In order to describe this model, we introduce the function $coopt(a, seed)$, that expresses that a device $a$ has been co-opted by the entity knowing *seed*, and is able to prove it. The community $\Lambda$ based on *seed* is then defined by:

$$\Lambda = \{a \mid coopt(a, seed)\}$$

The global consistency of the community is ensured since we have:

$$\forall a \in \Lambda, \Lambda(a) = \{b \mid coopt(b, seed)\}$$

That is, a device $a$ trusts any device $b$ able to prove it has been co-opted by the entity knowing *seed*.

### 3.2.2 Examples

Instances of the preshared derived data model can be classified in two categories. Some of them require an on-line entity knowing *seed* at the time of the communication, in order to mutually authenticate the two devices (i.e. to ensure both of them have been co-opted). In Kerberos [13], the seed information is a list of (device, key) pairs stored by the Kerberos server. Each device is co-opted by the server through the insertion in the device of a key it shares with the server and the update of the server list. In [18], Perrig et al. propose to secure sensor networks by using a key setup corresponding to the on-line preshared derived data model. In their proposal, a group of sensor nodes are linked to a base station. Each node shares a symmetric key with the base station in order to secure the communications between them. When two nodes want to communicate securely, they use the relation each of them have with the base station to exchange a session key.

The second category of instances of the preshared derived data model is based on off-line verifiable mechanisms: two devices do not need the entity knowing *seed* to be available when they want to securely communicate. Issuing certificates for the members of the community is a classical example of off-line verification: each member of the community knows and trusts the public key of the central authority, and holds a certificate verifiable with it for its own public key. Consequently, it can prove it belongs to the community and check the certificates presented by the devices claiming so.

### 3.2.3 Limitations

By opposition to the preshared common data model, both categories (on-line and off-line) of preshared derived data model enable the banishment of a device from the community: this device is erased on the central on-line server, or its certificate is revoked by the central off-line server.

However, the preshared derived data model fails in offering Requirements 1 & 2. First, Requirement 1 makes the on-line verifiable preshared derived data approach unsuited. Second, both approaches do not fulfill the Requirement 2, since the entity knowing *seed* must be available to co-opt the new device when the network evolves.

## 3.3 Delegation model

### 3.3.1 Scheme

The preshared derived data model has been extended with delegation in order to reduce the limitations related to its centralization. In the preshared derived data model, two entities belong to the same community if they have been co-opted by a third one they both trust. In the delegation model, each device that has been co-opted can itself co-opt other devices. In other words, the delegation model considers the co-option relation as transitive. Using the previous notation, we have:

$$\forall a, b, c, \; coopt(a, b) \wedge coopt(b, c) \Rightarrow coopt(a, c)$$

where $a$, $b$ and $c$ are devices, and $coopt(a, b)$ defines that $a$ has been co-opted by $b$[3]. Given this transitive relation, in a community, there exists at least one root device $r$ such as:

$$\forall a \in \Lambda, coopt(a, r)$$

In other words, the community $\Lambda$ is defined by:

[3]For notation simplicity, we combine a device and its representative information.

$$\Lambda = \{a \mid coopt(a, r)\}$$

The insertion of a new device $b$ does not require the device $r$ since it can be co-opted by any present device that already belongs to the community.

Locally, a device $a$ considers any device $b$ able to prove that it has been co-opted by $r$, possibly transitively, as belonging to its community:

$$\forall a \in \Lambda, \Lambda(a) = \{b \mid coopt(b, r)\}$$

#### 3.3.2 Examples

There are two kinds of implementation of the delegation model: in one hand, the authority can be centralized, in the sense that it is unique. All the devices trust the same central authority $r$, and a device $a$ is trustable if there is a chain of co-option linking it to $r$, i.e., $a$ is able to prove that it has been co-opted by $r$, possibly transitively.

By opposition to this centralized approach, the PGP-like approach is more distributed. In the PGP model [24], there is not a unique root authority that everyone trusts. Each user chooses the public keys she or he trusts as associated to their claimed owners, and inserts them in her or his local public key repository. He or she can also issue certificates for these public keys using her or his own private key. By sharing certificates, users create and maintain trust networks used to authenticate each other and to communicate securely with people they may never have met before.

Practically, in PGP, the certificates are currently stored in supposedly always available on-line servers. Hubaux et al. [11] have adapted the PGP model to ad hoc networks constraints. In their proposal, each device maintains a local repository containing a limited number of certificates. Because of interesting properties of the systems, having only a small number of certificates in each device is sufficient in most cases to make a trust relation between any two of them. Eschenauer et al. [8] also propose to use this kind of model to establish trust relations between devices following different policies in ad hoc networks.

#### 3.3.3 Limitations

Compared to the preshared derived data model, the delegation model allows community evolution through any device that has already been co-opted.

In the centralized delegation model, all the devices share the same $r$. Two devices $a$ and $b$ that want to check if they belong to the same community will check if they both have been transitively co-opted by $r$. In this case, the main problem is to choose the root of the hierarchy $r$ (i.e. the trusted authority), to set it up and to maintain it. If $r$ has to be removed from the community, the chains of co-option will no longer lead to a valid device, and the community is broken. Consequently, $r$ is mandatory, which does not meet Requirement 2.

In the decentralized model, each device is its own root and delegates its trust to other devices. However, this does not ensure that the trust relation is mutual: a given device $a$ that trusts $b$ (i.e., $a$ has co-opted $b$) may not be trusted by $b$. Given this fact, the decentralized model does not allow the use of any device of the community to insert a new community: in this example, a device inserted by $a$ will not be trusted by $b$. Consequently, the delegation model does not comply with Requirements 2 and 4.

### 3.4 Resurrecting Duckling model

#### 3.4.1 Scheme

The Resurrecting Duckling model [22] ensures the security of transient point-to-point relations between devices in ad hoc networks. In order to build a secure relation between two devices, one of them imprints the other, building a master-slave relation: they exchange cryptographic material using a supposedly secure channel (e.g. a physical direct connection) to ensure authenticity and confidentiality of the communication. When the master device imprints the slave, it co-opts it and is implicitly co-opted by it: the slave trusts the master, and the master also implicitly trusts the slave. The imprinting operation is a symmetric co-option operation:

$$\forall a, b \; coopt(a, b) \Leftrightarrow coopt(b, a)$$

However, this relation is not supposed to be transitive:

$$\forall a, b, c \; coopt(a, b) \wedge coopt(b, c) \not\Rightarrow coopt(a, c)$$

The imprinted device remains linked to its imprinter until the latter breaks the relation between them. At this time, it gets back to the imprintable state.

#### 3.4.2 Examples

Some recent works [3, 6, 9, 21] use and extend the Resurrecting Duckling model.

In [21], Stajano proposes to manage more than one-to-one relations by enabling the master device to upload security policies while imprinting the slave. These policies can, for example, specify the list of devices authorized to perform certain actions on the slave device. This extension offers a form of direct co-option from master to slave: a master device co-opts a set of other devices for this specific slave when imprinting it. However, this co-option exists only at the time of the imprinting. No mechanism is proposed to manage a dynamic set of devices that would have to evolve after imprinting.

Based on [3], Capkun et al. recently proposed a technique [6] to set up security associations (i.e. to exchange certificates securely) between devices in a fully decentralized ad hoc network. In their proposal, each device is its own authority domain. There is no need for a central authority, nor central PKI, nor trusted third party. When two users meet and want to communicate, they use a "privileged side channel" *à la* Resurrecting Duckling to establish their relation securely. Capkun et al. also introduce the "friend" relation. Two devices are friends if they have already established a security association and trust each other as a reliable source of information for providing certificates. A given device can ask a friend for certificates about some other devices it has no certificate for and wants to communicate securely with. This friend relation is explicitly non-transitive: a device does not consider a friend's friend as a friend.

#### 3.4.3 Limitations

The Resurrecting Duckling model is a generic model that needs to be adapted to specific circumstances.

The proposals [9, 21], dealing with more than two participants, consider exclusively transient relations. Consequently, they do not take into consideration long term communities and their evolutions, and do not address Requirement 3.

While [6] addresses secure point-to-point relations and efficient distribution of certificates, it does not address the problem of building a secure long-term community of devices. More particularly, the authors do not define how a

device chooses its "friend" devices. In a long-term community, this choice of "friends" would be very important. Moreover, no mechanism is proposed for the revocation of a friend or of a certificate. Consequently, this proposal does not address Requirement 3. Nevertheless, the "friend" concept is particularly interesting as it builds and distributes security relations easily with only light user involvement.

Table 1 summarizes the requirements fulfilled or not by each model.

# 4. A NEW APPROACH TO SECURE LONG TERM COMMUNITIES

We now propose a distributed approach that fulfills Requirements 1, 2, 3 & 4 by building symmetric and transitive long term trust relations.

## 4.1 Scheme

In our proposal, like in [6], there is no central information nor central element in the community: each device considers itself as the central element of its community, around which the whole community evolves. It manages its own knowledge of the community, and shares information with the other devices to keep this knowledge accurate.

Each device must be able to identify and authenticate the other devices that belong to its community and communicate securely with them. For this purpose, each device manages its own provable identity. A provable identity is an identity that anyone can check, although being very hard to impersonate. For instance, the public key of a public/private key pair is a provable identity: a device pretending being identified by its public key can prove it by signing a challenge with its private key. It will also be the only one able to decrypt a message encrypted with its identity, i.e. its public key. SUCV [15] and CAM [16] are other mechanisms based on the idea of provable identity that have already been used to secure roaming in IP networks [2], to bootstrap secure routing [5] in IP ad hoc networks, or to make node addresses verifiable in a fully self-organized ad hoc network [6].

Using their provable identities, two devices can create a point to point secure channel by encrypting their messages with the public key corresponding to the identity of the recipient and signing it with the private key of the source. For performance purpose, provable identities can be used to exchange a symmetric session key.

The main problem we have to solve is to enable each device to securely manage the provable identities of the other members of its community. In our proposal, each device $a$ manages the knowledge $\Lambda(a)$ it has of its community $\Lambda$. Two devices $a$ and $b$ have to respectively know that each other belongs to its own community to communicate freely: $a$ must know $b$ as being a member of its community (i.e. $b \in \Lambda(a)$), and $b$ must know $a$ as a member of its community (i.e. $a \in \Lambda(b)$).

Locally, a device $a$ can consider its trust relation with another device $b$, that it knows as being or having been in its community, in three different states:

**Mutual trust** : $b$ belongs to $\Lambda$ and has already been met by $a$: $a$ trusts $b$, $b$ trusts $a$, and $a$ knows that $b$ trusts $a$.

**Unilateral trust** : $b$ belongs to $\Lambda$, but $a$ never met $b$. $b$ has been introduced to $a$ by $c$, $c \in \Lambda(a)$. Locally, $a$ trusts $b$, but does not know if $b$ knows that $a$ belongs to the community, and consequently if it trusts $a$.

**Distrust** : this state is equivalent to the issue of revocation for a certificate. $b$, while being locally known by $a$, does not belong to $\Lambda(a)$. $b$ was formerly in $\Lambda$, but does not belong to it anymore, because it has been banished or removed.

The mutual trust relation between two devices is involved by their symmetric co-option. Locally, a device $a$ has a proof that $b$ trusts $a$, i.e., $coopt(a, b)$ for every device $b$ it has a mutual trust relation with.

Every device $c$ that $a$ knows as unilaterally trusted has been introduced to it by a device $b$, $c$ being either mutually trusted by $b$ or unilaterally trusted by $b$. For each device $c$, $a$ has a chain of co-option provided by $b$ that proves that $c$ trusts $a$. When $a$ will meet $c$, it will be able to provide this chain of co-option to prove to $c$ that it should trust $a$.

The distrusted devices are those that $a$ explicitly distrusts. $a$ will not accept any proof that they belong to the same community.

A device $a$ believes that another device $b$ belongs to its community if $b$ is known as mutually trusted or unilaterally trusted, without $b$ having been explicitly removed or banished. Using our notation, and $\mathrm{MT}(a)$ and $\mathrm{UT}(a)$ being respectively the sets of devices $a$ has a mutual trust relation with, or a unilateral trust relation with, we have:

$$\Lambda(a) = \mathrm{MT}(a) \cup \mathrm{UT}(a)$$

If $\mathrm{DT}(a)$ is the set of devices $a$ distrusts, i.e., that have been explicitly removed or banished, $a$ will use this set to inform the other devices of its community of the distrusted devices. By construction, the sets of devices are totally disjoint:

$$\forall a \in \Lambda,\ \mathrm{MT}(a) \cap \mathrm{UT}(a) = \emptyset,$$
$$\mathrm{MT}(a) \cap \mathrm{DT}(a) = \emptyset,$$
$$\mathrm{UT}(a) \cap \mathrm{DT}(a) = \emptyset$$

## 4.2 Secure evolution of the community

$a$ being a device in $\Lambda$, any modification of $\Lambda(a)$ may not be due to a real evolution of $\Lambda$: $\Lambda(a)$ can evolve because a device $c \in \Lambda(a)$ informed $a$ that $\Lambda$ has been modified. In this case, the modification of $\Lambda(a)$ is just a synchronization of $a$'s knowledge with $c$'s one. Synchronization is described in §4.3. Here, we consider the case of a real $\Lambda$ evolution, that is when $\Lambda(a)$ evolves and $a$ is the first device in $\Lambda$ where the modification occurs.

The evolution of the community is locally initiated by the authority (i.e., the user): she or he is the one that informs the local device that a new device has to be inserted, removed of banished. This is the only time the user is involved. After that, the information will be forwarded from $a$ to the other devices belonging to $\Lambda(a)$ (see §4.3).

Requests for evolution being security-relevant, the device $a$ on which the action is performed has to authenticate the authority, i.e. the user. This authentication is strictly local to each device. Consequently, devices of the community can use mechanisms that are different, and each device can use the best suited to itself. Moreover, because the devices of a given community do not share a unique representation of the authority, our proposal is not centralized around such a representation, but is really distributed.

### 4.2.1 Initialization

| | Preshared Common Data | Preshared Derived Data (on-line) | Preshared Derived Data (off-line) | Delegation Model (central.) | Delegation Model (PGP-like) | Resurrecting Duckling Model |
|---|---|---|---|---|---|---|
| Req. 1 | √ | × | √ | √ | √ | √ |
| Req. 2 | √ | × | × | × | × | √ |
| Req. 3 | × | √ | √ | √ | √ | × |
| Req. 4 | √ | √ | √ | √ | × | × |

**Table 1: The requirements fulfilled by some current models**

In its initial state, a device $a$ is the only member of its community. During the initialization phase, the device inserts itself in $\mathrm{MT}(a)$. At this moment,

$$\Lambda(a) = \Lambda = \{a\}$$

$a$'s knowledge of $\Lambda$ is coherent and valid from a security point of view, because $a$ only considers devices that really are in its own community (here, itself) as so.

#### 4.2.2 Insertion

As seen in §4.1, a device $a$ will only communicate with devices in $\Lambda(a)$. Thus, a user who inserts $a$ in another device $b$'s community also has to insert $b$ in $a$'s, otherwise $a$ will not communicate with $b$. As a consequence, the trust relation set up between them at the insertion time has to be symmetric[4]:

$$\forall a, b \; coopt(a, b) \Leftrightarrow coopt(b, a)$$

that is, when $a$ co-opts $b$ and inserts it in its community, $b$ also trusts and co-opts $a$, i.e., $b$ inserts $a$ in its community. Because they both know they are being inserted by the other one, each of them inserts the other as a mutually trusted device:

$$\mathrm{MT}(a) := \mathrm{MT}(a) \cup \{b\} \text{ and } \mathrm{MT}(b) := \mathrm{MT}(b) \cup \{a\}$$

Moreover, $a$ (resp. $b$) issues a ticket to $b$ (resp. $a$) that proves $coopt(b, a)$ (resp. $coopt(a, b)$).

One may argue that inserting manually the provable identity of a device in the other is not a user-friendly approach. A first way to solve this problem would be to use a secure side channel to transmit the provable identities as described in [6].

Another mean would be to use user-friendly representations of provable identities based for instance on random art representation [17]. When two devices $a$ and $b$ have to insert each other in their respective communities, each of them broadcasts its own provable identity, and collects all the provable identities it can receive. After a short time, $a$ and $b$ display the random art representation of their respective provable identities and of all the provable identities they have collected. By comparing the displays on both devices, the user can easily choose the right ones. Because of the interesting properties of provable identities, this mechanism can be used on a channel where attackers can both eavesdrop and insert messages.

It would be clearly impractical for the user to co-opt $a$ manually on all the devices in $\Lambda$. The synchronization phase (explained in §4.3), that occurs after two devices $a$ and $b$ co-opted each other, frees the user from this.

#### 4.2.3 Removal

The removal of a device $b$ can be done on $b$ itself, because $b$ is available at the moment of the removal (see §2.3.3). $b$ should first inform devices in $\Delta(b)$ of its removal, that is:

$$a \in \Delta(b),\ \mathrm{MT}(a) := \mathrm{MT}(a) \setminus \{b\},$$
$$\mathrm{UT}(a) := \mathrm{UT}(a) \setminus \{b\},$$
$$\mathrm{DT}(a) := \mathrm{DT}(a) \cup \{b\}$$

Then, $a$ will inform the other devices that $b$ has been removed when they will synchronize (see §4.3). $b$ then clears its local information:

$$\mathrm{DT}(b) := \mathrm{DT}(b) \cup \mathrm{UT}(b) \cup \mathrm{MT}(b) \setminus \{b\}$$
$$\mathrm{MT}(b) := \{b\},\ \ \mathrm{UT}(b) := \emptyset$$

and consequently, we have:

$$\Lambda'(b) := \{b\}$$

If no device is available during $b$'s removal, i.e. $\Delta(b) = \emptyset$ at the time of the removal, then no remaining member of the community will be warned that $b$ does not belong to the community anymore. However, because $b$ does not trust any device from the community after its removal, it will not accept to communicate with them, and the security is anyway ensured. For more security, a user could inform one of the devices still belonging to the community that $b$ does not belong to it anymore, using the thereafter explained banishment mechanism.

#### 4.2.4 Banishment

The banishment of a device $b$ from the community $\Lambda$ is made from any device $a \in \Lambda$ such as $b \in \Lambda(a)$ ($a$ does not need to be in $\Lambda(b)$). On $a$, the user simply has to declare that $b$ is banished: $a$ removes $b$ from its community by removing it from $\mathrm{MT}(a)$ or $\mathrm{UT}(a)$, and inserts it in $\mathrm{DT}(a)$:

$$\mathrm{MT}(a) := \mathrm{MT}(a) \setminus \{b\},$$
$$\mathrm{UT}(a) := \mathrm{UT}(a) \setminus \{b\},$$
$$\mathrm{DT}(a) := \mathrm{DT}(a) \cup \{b\}$$

and consequently, we have:

$$\Lambda(a) := \Lambda(a) \setminus \{b\}$$

The devices in $\Lambda(a)$ will be informed that $b$ does not belong to $\Lambda$ anymore during the synchronization phase, described in §4.3.

One can notice that the banishment operation can also be used for removal.

### 4.3 Distributed loose consistency

When devices of the same community cannot communicate, they cannot update their respective knowledge of the community. Consequently, our proposal cannot ensure the global consistency of the community.

However, in order for the community to stay consistent across its evolutions, each member exchanges information with the others when they are mutually reachable. By this synchronization operation, we ensure the loose consistency of the community.

At the community level, a device can be unknown, trusted, or distrusted: a device is unknown before its insertion; it becomes trusted the first time it is inserted in the community

[4]We should notice that, during the insertion, $a$ and $b$ have to be mutually reachable (i.e. $a \in \Phi(b)$ and $b \in \Phi(a)$), *à la* Resurrecting Duckling model.

(i.e., when at least one device of the community mutually trusts it); finally, it becomes distrusted the first time a device in the community marks it as distrusted[5]. Consequently, these states are strictly timely ordered: the device $a \in \Lambda$ knowing a device $b$ in its most advanced state should be considered as the most up-to-date, and all the other devices of the community should be synchronized according to $a$'s knowledge.

In order to ensure the loose consistency, a device trusts the other devices in its community to co-opt new devices in it, and to provide information about it. Thus, the co-option relation is transitive:

$$\forall a, b, c,\ coopt(a, b) \wedge coopt(b, c) \Rightarrow coopt(a, c)$$

that is, $a$, $b$ and $c$ being three devices, if $c$ trusts $b$ as being in its community $\Lambda$, and $b$ trusts $a$ as being in $\Lambda$, then $c$ trusts $a$ as being in $\Lambda$.

Using the "friend" principle described in [6], we can express this fact as such: $a$ being a device in $\Lambda$, $\forall b \in \Lambda(a)$, $a$ considers $b$ as a friend. However, unlike [6] that do not accept trust transitivity, we accept a chain of co-option as being a proof of belonging. In our proposal, all the devices belonging to the same long term community are submitted to the same policy and trust each others. When $a$ inserts $b$ in the long term community, this insertion is valid for all the devices that believe $a$ to be in their community: any device that trusts $a$ to issue proofs of co-option will then trust $b$ the same way to insert new members in the community.

#### 4.3.1 Synchronization between mutually trusted devices

The synchronization of two devices $a$ and $b$ that know each other as mutually trusted always follows the same algorithm. First of all, $a$ and $b$ exchange their respective information about the devices they know as distrusted, and each of them inserts the devices it did not know as distrusted in its own DT. We have:

$$\forall c \in \mathrm{DT}(a) \wedge c \notin \mathrm{DT}(b)\ ,\ \mathrm{DT}(b) := \mathrm{DT}(b) \cup \{c\},$$
$$\mathrm{MT}(b) := \mathrm{MT}(b) \setminus \{c\},$$
$$\mathrm{UT}(b) := \mathrm{UT}(b) \setminus \{c\}$$
$$\forall d \in \mathrm{DT}(b) \wedge d \notin \mathrm{DT}(a)\ ,\ \mathrm{DT}(a) := \mathrm{DT}(a) \cup \{d\},$$
$$\mathrm{MT}(a) := \mathrm{MT}(a) \setminus \{d\},$$
$$\mathrm{UT}(a) := \mathrm{UT}(a) \setminus \{d\}$$

$a$ and $b$ then compare the devices they know as mutually trusted. $b$ (resp. a) inserts each device $c$ (resp. d) known by $a$ (resp. b) as mutually trusted but that $b$ (resp. a) does not know:

$$\forall c \in \mathrm{MT}(a) \wedge c \notin \mathrm{MT}(b) \cup \mathrm{UT}(b),$$
$$\mathrm{UT}(b) := \mathrm{UT}(b) \cup \{c\}$$
$$\forall d \in \mathrm{MT}(b) \wedge d \notin \mathrm{MT}(a) \cup \mathrm{UT}(a),$$
$$\mathrm{UT}(a) := \mathrm{UT}(a) \cup \{d\}$$

Finally, they exchange the devices they know as unilaterally trusted:

$$\forall c \in \mathrm{UT}(a) \wedge c \notin \mathrm{MT}(b) \cup \mathrm{UT}(b),$$
$$\mathrm{UT}(b) := \mathrm{UT}(b) \cup \{c\}$$
$$\forall d \in \mathrm{UT}(b) \wedge d \notin \mathrm{MT}(a) \cup \mathrm{UT}(a),$$
$$\mathrm{UT}(a) := \mathrm{UT}(a) \cup \{d\}$$

Then, $a$ (resp. $b$) exchanges all the tickets that $a$ (resp. $b$) previously received from the devices belonging to $\mathrm{MT}(a)$ (resp. $\mathrm{MT}(b)$), and that do not carry on distrusted devices.

Thanks to the transitive property of the co-option relation, when $a$ (resp. $b$) will meet a device $c$ with which it has an unilateral trust, it can provide to $c$ all the tickets proving that $c$ should trust $a$ (resp. $b$). For example, if $d$ previously met $c$, $d$ has issued to $c$ a ticket proving that $coopt(c, d)$. When $c$ meets $b$, $d$ is included in $\mathrm{UT}(b)$, and $b$ receives the tickets that prove that $coopt(b, c)$ and $coopt(c, d)$. When $b$ meets $a$, $c$ and $d$ are included in $\mathrm{UT}(a)$, and $a$ receives the tickets to prove that $coopt(a, b)$, $coopt(b, c)$ and $coop(c, d)$. Then, when $a$ meets $c$ or $d$, it will be able to provide the tickets informing $c$ or $d$ that they should trust $a$.

A device $a$ has to initiate a synchronization each time $\Delta(a)$ is modified, either because $\Phi(a)$ has evolved, or because $\Lambda(a)$ has evolved.

#### 4.3.2 Evolution of $\Phi$

First of all, nothing is to be done when $\Delta(a)$ evolves because a device $b \in \Lambda(a)$ left $\Phi(a)$, because this event cannot be predicted.

$\Delta(a)$ can also evolve because $b \in \Lambda(a)$ entered in $\Phi(a)$. Two cases are possible: either (1) $a$ knows $b$ as unilaterally trusted, or (2) it knows $b$ as mutually trusted.

If $b \in \mathrm{UT}(a)$ (case 1), $a$ first has to provide to $b$ the tickets proving to $b$ that it is trustable, i.e., there exists a chain of co-option starting from $b$ and leading to $a$. $b$ checks these tickets, both $a$ and $b$ mark each other locally as mutually trusted, and they exchange the tickets proving they trust one another. Using our notation, we have:

$$\mathrm{UT}(a) := \mathrm{UT}(a) \setminus \{b\},$$
$$\mathrm{MT}(a) := \mathrm{MT}(a) \cup \{b\},$$
$$\mathrm{MT}(b) := \mathrm{MT}(b) \cup \{a\}$$

Now that $a \in \mathrm{MT}(b)$ and $b \in \mathrm{MT}(a)$, they can synchronize their knowledge of the community as described in §4.3.1.

In the case where the chain of co-option provided by a device $a$ involves a proof of co-option issued by a device once in the community but now distrusted, and if $a$ is not able to provide other valid proofs of co-option, $a$ should not be inserted, for obvious security reasons. Nevertheless, it could be possible that $a$ should indeed still belong to the community. For example, $a$ could have been legitimately inserted in the community by a device $e$ that have been banished before $a$ could meet any other device of the community. Locally, and for any unexplained (good) reason, $a$ has not marked $e$ as distrusted yet. In this case, $a$ is unknown by the other members of the community. So, the user(s) will simply have to enter $a$ explicitly in the community, using any other device $f$ still belonging to it. When $a$ and $f$ will synchronize their knowledge of the community, $f$ will inform $a$ that $e$ is now distrusted, and $a$'s knowledge of the community will consequently be up to date.

If $b \in \mathrm{MT}(a)$ (case 2), $a \in \mathrm{MT}(b)$: $a$ and $b$ know each other, and have already met at least once. During the period $b$ was not in $\Phi(a)$ (and consequently $a$ was not in $\Phi(b)$), the respective knowledge $\Lambda(a)$ and $\Lambda(b)$ may have evolved. Consequently, $a$ and $b$ may have to synchronize (see §4.3.1).

Due to parallel evolutions of the network, a device $a$ could get able to communicate with a device $b$ (i.e., $b \in \Phi(a)$) it knows as mutually trusted or unilaterally trusted, that in fact have banished it. In this case, while $a$ trusts $b$, $b$ does not trust $a$ anymore. Consequently, it will not reply to its requests for communication. In particularly talkative implementations, $b$ could send $a$ messages informing it has been banished.

[5] If a device have been erroneously banished or removed, its user will have to reset it, causing the device to have a new provable identity, and insert it again.

### 4.3.3 Evolution of $\Lambda$

$\Delta(a)$ can also evolve when a new device $b$ enters $\Lambda(a)$. We already detailed the insertion phase in §4.2.2 and the synchronization between the devices $a$ and $b$ (see §4.3.1). In order to guarantee the loose consistency, $a$ and $b$ also inform the devices in $\Delta(a) \cup \Delta(b)$ of the modification of the community by exchanging security relevant information (i.e., tickets) with these devices.

More specifically, if $c \in \Delta(a)$ (resp. $d \in \Delta(b)$), the synchronization between $a$ and $c$ (resp. $b$ and $d$) consists in synchronizing $a$ and $c$ (resp. $b$ and $d$) as if $c$ (resp. $d$) just entered $\Phi(a)$ (resp. $\Phi(b)$) as described in §4.3.2.

$\Lambda(a)$ can also evolve because $a$ has inserted a device in $\mathrm{DT}(a)$. In this case, it synchronizes with the devices in $\Delta(a)$, as defined in §4.3.1.

### 4.3.4 Loose consistency limitations

The loose consistency mechanism presents limitations in very particular cases. As an example, we can take the case of the merge of two communities $\Lambda(a)$ and $\Lambda(b)$. $c$ and $d$ are two devices belonging respectively to $\Lambda(a)$ and $\Lambda(b)$, but that have been physically unreachable for a very long time. After $a$ and $b$ entered each other in their respective communities, $a$ informs $b$ that $c$ is in the community, and $b$ informs $a$ that $d$ is in the community during the synchronization phase. $a$ has never met $d$, and $b$ has never met $c$: $d \in \mathrm{UT}(a)$ and $c \in \mathrm{UT}(b)$ .

For $a$ and $b$, the four devices $a$, $b$, $c$ and $d$ belong to the same community. However, if $c$ and $d$ can communicate bi-directionally (i.e. $c$ in $\Phi(d)$) without first having synchronized their knowledge of the community with either $a$ or $b$, they will not know that they belong to the same community.

Nevertheless, since $c$ and $d$ do not know they belong to the same community, and according to Requirement 1 & 4, they are not allowed to communicate with each other. As soon as $c$ or $d$ will be in $\Phi(a)$ or $\Phi(b)$, it will update its knowledge, and will be able to prove to the other that they belong to the same community.

Alternatively, for punctuality purpose, the user can also make $c$ and $d$ to get into the same community by making them to mutually co-opt. After synchronization, the two devices will have a consistent knowledge of the community.

## 4.4 Implementation issues

We now provide hints about implementation.

About the cryptographic material, each device has to be able to prove its identity, to communicate securely with other devices, and to co-opt other devices. We propose to use a public/private key pair for each device, the public key (or its cryptographic digest) being its identity. Based on this public/private key pair, a device can be authenticated, exchange secret keys, and finally co-opt other devices by issuing certificates for their provable identity.

Our approach states that each device manages its own knowledge of the community. This can be implemented easily by managing three lists on each device: the first one contains the public keys of the mutually trusted devices, as well as the certificates proving each of them have co-opted the local device. The second list contains the public keys of the unilaterally trusted devices, as well as the chains of certificates that prove that they transitively trust the local device. The third list contains the public key of the distrusted devices. Synchronization between devices consists in securely exchanging messages about the local lists of each device.

Finally, in our approach, the user is only involved during the insertion, removal and banishment of a device. These operations being security-relevant, the user have to authenticate before any of them. The authentication process being strictly local to each device, each device is free to implement the best suited authentication mechanism.

# 5. CONCLUSION

We have presented a user-friendly distributed approach to set up and maintain a secure long term community over a SOHO or home ad hoc network. There is no central point to the community because each device of the community considers itself as the central point: any device can introduce any other in its community provided they can communicate, even over insecure links.

Our proposal ensures the required security services for the members of the community, while taking into account the dynamic nature and physical constraints of ad hoc networks: a community can physically split, evolve and merge while staying locally consistent for its members. Moreover, The role of the user is restricted to be the authority on each device of the community: she or he is involved only when the community evolves, i.e., when a device is inserted, removed or banished.

Consequently, our approach is particularly suited for home and SOHO networks that require to build long term communities of devices while requiring light user involvement in security mechanisms. Nevertheless, we strongly believe that our proposal could also be adapted to other fields of application having similar requirements, such as a relatively stable consortium that meets regularly.

We now plan to make some performance measurements of our proposal. More particularly, we want to test different rules for deciding when to trigger the synchronization phases. We are also thinking about trying different mechanisms for managing the chains of co-option. From another point of view, we have considered in our approach multiple authorities sharing the same policy. It would now be interesting to enable automatic security policy management in the community, taking into account the possibly conflicting interests between the authorities. Finally, we also plan to study how a device not belonging to the community could be temporarily integrated into it.

# 6. ACKNOWLEDGEMENT

The authors would like to thank the anonymous reviewers, as well as Jean-Louis Diascorn, Eric Diehl and Valérie Gayraud for their insightful remarks and comments.

This work have been partially funded by the French *Association Nationale pour la Recherche Technique*.

# 7. REFERENCES

[1] P. Albers, O. Camp, J.-M. Percher, B. Jouga, L. Mé, and R. Puttini. Security in Ad Hoc Networks: a General Intrusion Detection Architecture Enhancing Trust Based Approaches. In *Proceedings of the First International Workshop on Wireless Information Systems (WIS-2002)*, Apr. 2002.

[2] J. Arkko, J. Kempf, B. Sommerfeld, and B. Zill. IETF Draft: SEcure Neighbor Discovery (SEND), 2003.

[3] D. Balfanz, D. Smetters, P. Stewart, and H. Wong. Talking to strangers: Authentication in adhoc wireless networks. In *Proceedings of the ISOC Network and Distributed Systems Security Symposium*, Feb. 2002.

[4] G. Bell and J. Gemmell. A call for the home media network. *Communications of the ACM*, 45(7):71–75, 2002.

[5] R. B. Bobba, L. Eschenauer, V. Gligor, and W. Arbaugh. Bootstrapping security associations for routing in mobile ad-hoc networks. Technical report, University of Maryland, May 2002.

[6] S. Capkun, J. P. Hubaux, and L. Buttyan. Mobility helps security in ad hoc networks. In *Proceedings of the Fourth International Symposium on Mobile Ad Hoc Networking and Computing*, 2003.

[7] S. Corson and J. Macker. RFC 2501 : Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations, Jan. 1999.

[8] L. Eschenauer, V. D. Gligor, and J. Baras. On trust establishment in mobile ad-hoc networks. Technical report, University of Maryland, 2002.

[9] L. Feeney, B. Ahlgren, and A. Westerlund. Spontaneous networking: an application-oriented approach to ad hoc networking. *IEEE Communications Magazine*, June 2001.

[10] Y. Hu, A. Perrig, and D. Johnson. Ariadne: A secure on-demand routing protocol for ad hoc networks, 2002.

[11] J. P. Hubaux, L. Buttyan, and S. Capkun. The quest for security in mobile ad hoc networks. In *Proceedings of ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHOC)*, Long Beach, CA, October 2001.

[12] IEEE Standard Department. IEEE 802.11 Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY), 1999.

[13] J. Kohl and C. Neuman. RFC 1510 : The Kerberos Network Authentication Service (V5), 1993.

[14] Merriam-Webster. Merriam-webster online dictionnary, www.webster.com.

[15] C. Montenegro and C. Castelluccia. Statistically Unique and Cryptographically Verifiable (SUCV) identifiers and addresses. In *NDSS'02*, Feb. 2002.

[16] G. O'Shea and M. Roe. Child-proof authentication for mipv6 (cam). *ACM SIGCOMM Computer Communication Review*, 31(2):4–8, 2001.

[17] A. Perrig and D. Song. Hash visualization: a new technique to improve real-world security. In *International Workshop on Cryptographic Techniques and E-Commerce (CrypTEC '99)*, pages 131–138, 1999.

[18] A. Perrig, R. Szewczyk, V. Wen, D. E. Culler, and J. D. Tygar. SPINS: security protocols for sensor networks. In *Mobile Computing and Networking*, pages 189–199, 2001.

[19] M. K. Reiter. A secure group membership protocol. *IEEE Transactions on Software Engineering*, 22(1):31–42, Jan. 1996.

[20] S. W. Smith. Humans in the loop: Human-computer interaction and security. *IEEE Security & Privacy*, June 2003.

[21] F. Stajano. The Resurrecting Duckling – What Next? *Lecture Notes in Computer Science*, 2133:204–211, 2001.

[22] F. Stajano and R. Anderson. The resurrecting duckling: Security issues for ad-hoc wireless networks. In *7th International Workshop on Security Protocols*, pages 172–194, 1999.

[23] L. Zhou and Z. J. Haas. Securing ad hoc networks. *IEEE Network*, 13(6):24–30, 1999.

[24] P. Zimmerman. The PGP user's guide.