computing@tanet.edu.te.ua
www.tanet.edu.te.ua/computing

ISSN 1727-6209
International Scientific
Journal of Computing

SECURING DEVICES COMMUNITIES IN SPONTANEOUS NETWORKS

Nicolas Prigent [1], Christophe Bidan [2]

[1] Thomson R&D France, 1, avenue de belle fontaine, BP 19,
35511 Cesson-Sévigné Cedex, France, nicolas.prigent@thomson.net
[2] Supélec, Avenue de la Boulaie, BP 81127,
35511 Cesson-Sévigné Cedex, France, christophe.bidan@supelec.fr

Abstract: *We define a community as a set of devices able to communicate permanently or erratically and that share a long term trust relation. Small corporate networks or home networks are typical examples of such communities. Historically, the devices of the same community communicated over physically isolated wired networks. They are currently used over spontaneous networks, the characteristics of which have implications, in terms of their security and the mechanisms that can be used to protect such.*
In this article, we present a fully decentralized service of automated configuration of the security mechanisms dedicated to communities of devices that communicate over spontaneous networks. This service is located on each device of the community and manages information related to the environment of the device and to the security policy. Based on this information, it configures dynamically and automatically the security services available on the device to ensure its security and that of the community to which it belongs.

Keywords: – *Security, ad hoc networks, spontaneous networks, communities of devices, ubiquitous computing.*

1. INTRODUCTION

Networks have become very common. They are now available to small companies and private individuals, as in SOHO and home networks for instance.

Simultaneously to this wide adoption, ease of use has become a major concern in networks technologies. Auto-configuration is thus a key feature, and devices are programmed to set up their network automatically, as soon as they are interconnected.

Consequently, a lot of devices (computers, PDAs, home servers, etc.) have now auto-configuration network capabilities. This allows to create transiently networks anywhere anytime with any devices, and to exchange information about the services each of them propose without the users being aware of it [7]. Ad hoc and spontaneous networks are very representative of these new kinds of networks emphasizing auto-configuration.

While security is still an issue in automatically created networks, classical security mechanisms cannot be used to protect them without adaptation [12]. Indeed, they require extensive configurations that would be detrimental to ease of use. Moreover, they allow securing a set of known devices in a controlled and clearly defined environment, whereas automatically created networks are uncontrolled environments with fuzzy boundaries.

In this article, we focus on the security of communities. We define a community as a set of devices that are linked by a long term trust relation, and that communicate permanently or erratically over spontaneous networks. This long term trust relation is to be compared to the transient connectivity of the networks. The devices that belong to the same corporate network or to the same home network are examples of such communities.

We present a fully distributed service for automated configuration of security services in communities. This service is located on each device of the community. It allows managing information related to the environment of the device and to the security policy. Based on this information, it configures dynamically and automatically the locally available security services to ensure its security and that of the community to which it belongs.

In chapter 2, we provide a more precise notion of devices community. In chapter 3, we introduce the properties of spontaneous networks. In chapter 4, we describe the security objectives. In chapter 5, we present the mechanism we designed to securely manage the group of devices that belong to the community. Finally, we present the service of automated configuration in chapter 6.

2. DEVICES COMMUNITIES

In this article, we deal with the automated configuration of security services for communities of devices in spontaneous networks.

We herein identify "*device*" as any entity that can work autonomously on the network and that has reasonable computation power. Desktop and laptop computers, PDAs and mobile phones are typical examples of devices. More specifically, a device does not require any other device to work and communicate on the networks. Consequently, we do not consider entities such as wireless keyboards and mice, or removable storage (hard drive, passive MP3 players, etc.) as devices, but consider them as *peripherals*. A peripheral does not communicate on the network autonomously: it is connected to another given device in a point-to-point manner, even if this device may change with time. Several proposals have been published on the security of the point-to-point communications between a device and its peripherals when they communicate over insecure channels [1,14]. Consequently, these aspects will not be studied more extensively in this article.

We identify "*community*" as a set of devices that share a long term trust relation and that are able to communicate permanently or erratically. The trust relation between the devices of a given community is a *long-term* trust relation, because when a device belongs to a community, it does so for an *a priori* long time. It leaves it *a priori* definitively when it is given, sold, lost, broken or stolen. The set of computers and PDAs of a small company is a typical example of a devices community. The trust relation between the devices of such a network is due to the fact that they belong to the same company. Similarly, the set of devices that form the home network of a family is a devices community.

In this article, we make the reasonable assumption that the users of the devices that belong to the same community share the same interest in the fact that this community is secure, and have no interest in attacking it. Consequently, unless it has been compromised and is no more under the control of its legitimate user, a device belonging to a community behaves legitimately.

3. SPONTANEOUS NETWORKS

Historically, the devices that shared a trust relation (and, according to our terminology, that formed a community) were connected to the same physically isolated network and thus was considered secure. The trust was deduced from the ability to physically access the network. Consequently, there was a very strong link between the community and the physical network within which it existed.

Devices communities are nowadays used on top of *spontaneous networks*. A spontaneous network consists of a set of devices that may not all belong to the same community and are networked transiently for collaborative activities [6]. This *short term* connectivity is to be considered, relative to the *long term* trust relation that exists between devices in the same community. Spontaneous networks are intended to be user-friendly: their goal is to enable spontaneous collaboration between devices, and therefore the establishment of such a network should not require lengthy configurations steps.

Spontaneous networks are similar to ad hoc networks [5], with which they share some properties. First, a spontaneous network can be created anywhere and with any set of devices without relying on any infrastructure. Thus, the properties and environment of the devices change with time. For instance, the IP address of a device that connects to various links will change according to the link to which it is currently connected and to the addresses that are already assigned. Consequently, it is not possible to use the transient properties of a device (such as its IP address or the link to which it is connected for instance) in the definition of the trust relation.

Moreover, due to the dynamic topology of spontaneous networks, devices may join and leave a given spontaneous network arbitrarily. Thus, a spontaneous network may contain devices that belong to different communities, and there is no link anymore between the fact that a device is connected to the network and the fact that it is trustworthy.

A community can also be physically split in more than one partition, possibly as many as there are devices: the devices of a given community can be connected to different spontaneous networks. As an illustration, we can consider the case of a community made of 4 laptops (represented in black circles on Fig. 1) that belong to a small company. When two employees go out with their laptops for a meeting outside the company, two partitions appear. One is made of the two laptops brought for the meeting (this partition may also contain devices belonging to the visited company), the other of the two laptops that stayed in the head office. From a security point of view, this characteristic leads to the fact that one cannot suppose an always available central entity or any physically controlled network link.

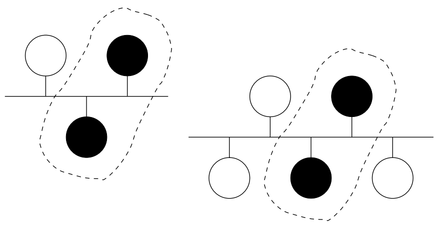

Fig. 1 – A community partitioned over two spontaneous networks.

The above characteristics must be taken into account in the design of the security mechanisms that will be used to protect devices communities communicating over spontaneous networks. Moreover, because spontaneous networks must be user friendly, priority must be given to automated configuration in the context of the security mechanisms.

4. SECURITY REQUIREMENTS

The *fortress model* is the traditional approach to protect a network made of entities linked by a trust relation. First, it involves defining the boundary of the network, i.e. in marking the difference between the “inside” entities that must be protected (in our case, the devices that belong to the community) and the “outside” entities from which the “inside” entities have to be protected. Mechanisms then are set up to enforce this boundary. Of course, in the perspective of in-depth security, other mechanisms may be set up inside the boundary to more finely control user access to a particular service.

As shown in section 3, the medium cannot be used to define the boundary of a community in spontaneous networks: the fact that two devices can physically communicate does not mean that they belong to the same community. On Fig. 1 for example, devices represented by the white circles on the partition on the right are present to the same meeting as the ones represented by the black circles and are connected to the same link. While all these devices do not belong to the same community, they can physically communicate. They even have set up a spontaneous network intended to exchange information and services, and one device may collaborate with other devices that are connected to the same spontaneous network, but that do not belong to its established community. Due to this fact, different levels of access should be enforced, based upon whether a device communicates with another device that belongs to its community or with a device that does not belong to its community.

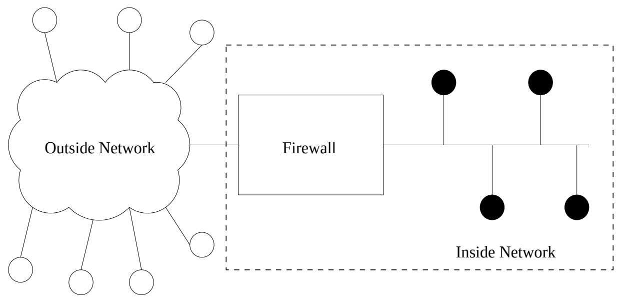

Fig. 2 – A firewall protecting physically isolated private network.

Network firewalls are frequently used to protect local area networks connected to the Internet (cf. Fig. 2). A network firewall has two main functionalities [4]. First, it physically marks the boundary between the network it protects (the “inside” network) and the uncontrolled network to which it is connected (the “outside” network). Then, it enforces the inside network security policy by filtering the cross-boundary communications. Messages that comply with the security policy are transmitted, while others are dropped.

In this respect, we believe that network firewalls are representative of the security functionalities that should be offered to the devices communities [13]. However, network firewalls are not adapted to the properties of spontaneous networks. Indeed, they are based on the hypothesis that all cross-boundary communication goes through them: the inside network is supposed to be physically protected, and an attacker can neither inject messages on it, nor eavesdrop. In the context of spontaneous networks, this hypothesis of the physically constrained network is not valid anymore, and one has to consider that an attacker may fully control the communication channels: he or she can eavesdrop, inject, or destroy any message on them.

In order to define a boundary, it is then first necessary to design a specific mechanism to allow the devices belonging to the same community to identify and authenticate one another. This mechanism must be user friendly, and user implication must be as limited as possible. More specifically, the mechanism must enable easy evolution of the community through insertion and/or removal of devices. It must also comply with the properties of spontaneous networks. As such, it must neither be based on any supposedly always available device, nor on any physically constrained medium. When an evolution occurs in the community while it is partitioned, devices of the community must get aware of this evolution when they can communicate again.

Once the boundary is defined, the cross-boundary exchanges must be controlled. Dealing with the example we used earlier (cf. Fig. 1), all the devices that take part in the meeting are supposed to interact, and some of them can offer some services (for instance, access to the local http or ftp services) to the other ones, even if they do not belong to the same community. However, a device does not have the same level of trust with the devices that are in the same spontaneous network but do not belong to its community than it has with the devices of its own community. As we make the hypothesis that the devices in the same community behave legitimately one with the other, outgoing communications initiated by a device that belongs to the community, and sent to a device that does not belong to it, are considered legitimate and should be authorized. By contrast, cross-boundary communications initiated from a device that does not belong to the community to a device that belong to the community must be controlled. While some services considered as "public" can be accessed by any device (should it belong to the community or not) other "private" services should be accessible only to the devices of the community. Thus, a communication from the outside is authorized only if the requested service is public.

In summary, protecting devices communities communicating over spontaneous networks requires:

- a mechanism that defines the boundary of the community by enabling the devices of the community to authenticate on another
- a mechanism that enforces the boundary by securing the communications between the devices of the community and controlling the cross-boundary interactions.

Because a community can be split over different spontaneous networks, we cannot assume the existence of an always available device that would act as a network firewall to protect all the devices of the community at one time. Consequently, each device may have to ensure itself its own security, particularly if it is the only device of its community on a particular spontaneous network. Due to this fact, a fully distributed solution has to be chosen. In section 5, we propose a fully distributed mechanism that allows definition of the boundary of the networks, i.e. to securely manage the group of devices belonging to the community and to distribute the keys among those devices. In section 6, we propose a mechanism that enforces this boundary.

5. SECURE MANAGEMENT OF THE COMMUNITY

In this section, we present the fully distributed mechanism we have designed to securely manage the group of devices that belong to the community. In this mechanism, there is neither central information nor central element: each device considers itself as the central element of its community, around which the whole community evolves. Each device manages its own knowledge of the current state of its community, and shares information with the other devices to keep this knowledge accurate.

Each device has a provable identity that is used for authentication with the other devices of the community. We call provable identity 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 is also the only one that can decrypt a message encrypted with its identity, i.e. its public key. SUCV [9] and CAM [10] are other mechanisms based on the concept of provable identity. In this paper, and without loss of generality, we consider that, like in [9,10], the provable identity of a device is the hash value of its public key.

To manage locally the knowledge of its own community, each device securely manages the provable identities of the other devices that belong to it or once belonged to it, but do not anymore. At the community level, a device can be *unknown*, *in*, or *removed*: a device is *unknown* while it has never been in the community. It becomes *in* when it is inserted in the community. Finally, it becomes *removed* when it is removed from the community. By hypothesis, we consider that a device that is removed from a community cannot be reinserted with the same provable identity. Consequently, these states are strictly timely ordered, and a device that once have been *in* will never be *unknown* again. Similarly, a *removed* device will never be *unknown* or *in* again.

Locally, each device maintains a local representation of the state of the other devices of its community. While a device does not explicitly keep information about the devices that are *unknown*, it maintains information about the devices that belongs to its community or once belonged to it using three mutually exclusive sets: *mutual trust*, *unilateral trust*, or *distrust*. For a device *a*, a device *b* is in set:

- *mutual trust* if *a* considers *b* as being in its community. Moreover, *a* has already communicated with *b* and knows that *b* also considers *a* as being in its community. *a* locally has the public key of *b*, and a proof that *b* considers *a* as being in its community.

- *unilateral trust* if *a* considers that *b* is in its community but *a* never communicated with *b*. As such, it does not know whether *b* knows that *a* is in its community. As will be shown later, *b* has been introduced to *a* by a device *c* that *a* knows as *mutually trusted*. *a* has also a proof that will be provided to *b* to prove to it that *b* should consider that *a* belongs to its community. *a* considers *b* as being in its community if it knows *b* in the state *mutual trust* or *unilateral trust*.
- *distrust* when *a* knows that *b* was in its community but does not belong to it anymore because it has been removed from it. This state is equivalent to the issue of revocation for a certificate: *a* explicitly distrusts *b* and will not accept any proof that they belong to the same community.

Originally, a device *a* is the only device in its community. During the *initialization* phase, *a* generates or obtains out of band a public/private key pair and an provable identity, inserts it in its set *mutual trust*, and sets the two other sets to void. At this time, *a*'s knowledge of its community 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.

The evolutions of the community are locally initiated on a device of the community by the authority (i.e., the user). For each evolution, the user informs a device *a* of the community that a new device is to be inserted or removed. This is the only time the user is involved. After that, the information will be forwarded by *a* to the other devices of the community. Requests for evolution being security-relevant, the device *a* on which the action is performed has to authenticate the authority. This authentication is strictly local to each device, and 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.

To *insert* a device *b* in the community of a device *a*, a user informs *a* that *b* now belongs to its community by providing to *a* the provable identity of *b* as the one of a new device of the community. The mechanism being fully decentralized, *a* and *b* both need to be informed of the insertion, and the user have to insert both *a* in *b*’s community, and *b* in *a*’s. As a consequence, the trust relation set up between them at the insertion time is symmetric. Because both *a* and *b* suppose that the user will properly insert it in the other one’s community, each of them inserts the other as a mutually trusted device. Moreover, *a* and *b* exchange their public keys, and *a* (resp. *b*) issues a ticket to *b* (resp. *a*) using its provable identity that proves that *a* (resp. *b*) considers that *b* (resp. *a*) belongs to its community.

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 [3]. Another mean is to use user-friendly representations of provable identities based for instance on random art representation [11]. 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 receives. After a short time, *a* and *b* display the random art representation of their respective provable identities and of all the provable identities they 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.

To *remove* a device *b* from a community, the user simply informs a device *a* of this community that this device is now *removed*. *a* then removes the provable identity of *b* from the set *unilateral trust* or from the set *mutual trust*, and inserts it the set *distrust*. Moreover, *a* deletes from its *unilateral trust* set each device *c* for which the chain of tickets that *a* owns contains a ticket emitted by *b*.

It would be *clearly* impractical for the user to inform manually all the devices of the evolutions of the community. After having considered the operations the user have to do to make the community evolves, we now present the way the devices exchange information to maintain the local knowledge of each of them up-to-date. The *synchronization of local knowledge* frees the user from this.

A device trusts the other devices in its community to provide information about it. In other words, it takes into consideration the information the other devices provide about the community, and update its own knowledge if another device that it knows in state *mutual trust* have some that are more up-to-date. As presented earlier in this section, the three states that a device may have with respect to a given community (*unkown, in, removed*) are strictly ordered. Due to this fact, and provided it has not been compromised, a device *a* of a given community that knows another device *b* in its most advanced state should be considered to have the most up-to-date knowledge of the community, and all the other devices of the community should be synchronized according to *a*'s knowledge.

For the community to stay consistent across its evolutions, each device *a* periodically sends to the other devices of its community a message containing the current knowledge it has of the community. More precisely, this message contains:

- The provable identities of the devices that *a* currently has in its *mutual trust* set, as well as the ticket issued by each of them that proves that it considers *a* as being in its community.
- The provable identities of the devices that *a* currently has in its *unilateral trust* set, as well as the chains of tickets that proves that each of them considers *a* as being in its community (the way these chains have been obtained are described later in this section).
- The provable identities of the devices that *a* currently has in its *distrust* set.

This message is authenticated by being signed with *a*'s public key.

When it receives such a message from a device it knows in its set *mutual trust*, a device *b* follows this algorithm: first of all, *b* checks the authenticity of the message. Then, it processes the information dealing with the devices contained in the *distrusted* set. For each of these devices *c* that *b* does not have in its own *distrust* set, *b* inserts *c* in its own *distrust* set and removes *c* if necessary from the set *unilateral trust* or *mutual trust*. Moreover, *b* deletes from its *unilateral trust* set each device *d* for which the chain of tickets that *b* owns contains a ticket emitted by *c*.

Then, *b* processes the information dealing with the devices contained in the *mutual trust* set. For each of these devices *c* that *b* does not have in its *mutual trust* set, *unilateral trust set* or *distrusted* set, *b* inserts *c* in its *unilateral trust* set and stores the ticket issued by *a* that proves that *a* considers that *b* belongs to its community. *b* will use this ticket during the next phase of introduction of self (this operation is explained later) to prove to *c* that *a* (that, by construction, *c* considers to be in its community) considers *b* as being in its community and so that *c* should then consider *b* as being in its community too. For each device *d* that *b* already has in its *unilateral trust* set, *b* checks if the chain of tickets it has for *d* is longer than one ticket. If it is the case, *b* replaces this chain of tickets, and stores a chain of tickets made of the single ticket issued by *a* for *b*.

Finally, *b* processes the information dealing with the devices contained in the *unilateral trust* set. For each device *c* that *b* does not have in its *mutual trust* set nor in its *distrust* set, and if *b* does not have *c* in its *unilateral trust* set, *b* checks that the chain of tickets contained in the message w.r.t. *c* is valid and does not contain a ticket issued by a device that *b* has in its *distrust* set. In this case, it stores *c* in its *unilateral trust* set, appends the ticket issued by *a* for *b* to the chain of tickets contained in the message, and stores the resulting chain as a proof that it will use to introduce itself to *c*. If *b* has *c* in its *unilateral trust* set, it checks whether the chain of tickets it currently has for it is longer than the one contained in the message. If it is the case, *b* appends the ticket issued by *a* for *b* to the chain of tickets contained in the message, and replaces by it the one he owned before for *c*.

When a device *b* known by a given device *a* as unilaterally trusted becomes reachable, *a introduces itself* to *b* in order for both of them to insert the other one in its own *mutual trust* set. *a* knows *b* as *unilaterally trusted* because, as we showed earlier, another device *c* that *a* knows as *mutually trusted* informed *a* during a synchronization of local knowledge that *b* also belongs to *a*'s community. During this synchronization of local knowledge, *c* also provided to *a* a chain of tickets that *a* can provide to *b* to prove that they belong to the same community. The introduction of self goes as follows: *a* sends to *b* a message of introduction of self that contains it provable identity, its public key, a ticket that proves that *a* considers *b* to be in its community, and the chain of tickets *a* has that proves to *b* that it should consider *a* as being in it community. When *b* receives such a message, it first checks that *a* is not in its *distrust* set, that the chain of tickets is valid, that it does not contain any ticket issued by a device that it has in its *distrust* set . In case of success, *b* inserts *a* in its *mutual trust* set and stores the ticket issued by *a* for *b* as well as *a*'s public key. *b* also generates a message that it sends to *a* and that contains its own public key and a ticket that proves that *b* now considers *a* as being in its community. When receiving this message, *a* stores both the public key of *b* and the ticket it emitted, removes *b* from its *unilateral trust* and inserts it in its *mutual trust* set.

After the operation of introduction of self, *a* and *b* now have each other in their *mutual trust* set and can then perform operations of synchronization of information.

6. ENFORCING THE BOUNDARY

After having presented in section 5 a mechanism to define and manage the boundary of communities, we present in this section a fully decentralized approach in which each device enforces this boundary in an autonomous way, while collaborating with the other devices to the security of the whole community. To that end, each device of the community embeds a service that monitors continuously the environment. According to it and to the security policy, the service dynamically configures a local message filter that enforces the security policy on the communications it takes part to.

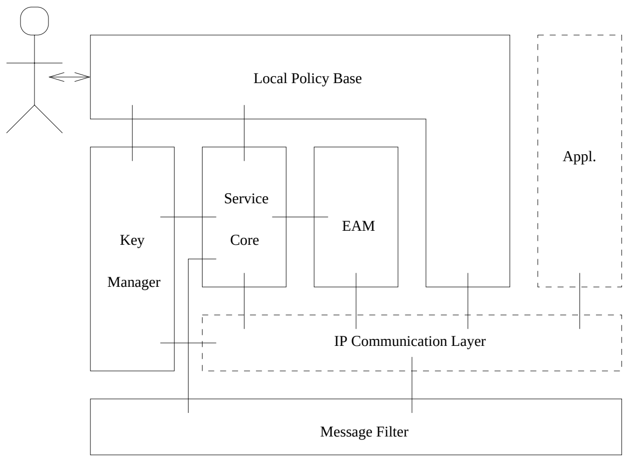

Fig. 3 – Architecture of the service.

The service (cf. Fig. 3) is made of five parts:

- The *Local Policy Base (LPB)*, that manages the information related to the security policy.
- The *Environment Awareness Module (EAM)*, that manages the information about the environment of the device.
- The *Key Manager*, that takes in charge authentication and key establishment with the other devices.
- The *Message Filter*, that enforces the policy on the communications in which the local device takes part.
- The *Service Core*, that generates the configuration rules for the message filter based on the information provided by the LKB, the EAM and the Key Manager.

The *Local Policy Base* is dedicated to the management of the information dealing with the security policy that is relevant to the configuration of the message filter. We can deduce the following abstract security policy from the security objectives we stated earlier:

- Communications are fully authorized between the devices belonging to the community. The channels being considered unsecure, these communications must be protected. If a specific fine-grained access control has to be done, we assume it will be at a higher layer.
- The devices of the community being trusted, they can access freely to the services that are offered by devices that do not belong to it.
- The access of devices that do not belong to the community to the services provided by the devices that belong to it must be controlled. Only public services are authorized.

Consequently the LPB has to manage at least two types of information:

- The set of the devices that belong to the community. This set is managed using the mechanism described in section 5, that furthermore provides the public key to authenticate each of them.
- The set of the services that are *public* and consequently can be accessed by any device, does it belong to the community or not. To make a service public, the user simply inserts the identifier of the service in the set of public services, and removes it from this set for the service to be no more public.

The LPB is the only part of the service that is accessible to the user. He or she has to act on it only when a modification occurs in the security policy.

The *Environment Awareness Module* manages information about the environment of the local device. To that end, the EAM of each device collaborates with the other EAMs available on the network, in a way similar to the advertisement and discovery mechanisms of Zeroconf [7]. First, the EAM advertises itself on the network by sending periodically to a pre-defined multicast IP address the link between its provable identity and its current IP address. Simultaneously, it listens on the same multicast address the advertisement made by the other devices that are available on the network, and updates the local knowledge of the identities of the available devices as well as their current IP address. The EAM does not ensure the legitimacy of the information it manages and provides. Particularly, an attacker may advertise an illegitimate link between a long term identity and an IP address, leading to an erroneous knowledge of the EAM. It is the role of the Key Manager to ensure devices authentications.

Communications channels being unsecure, the purpose of the *Key Manager* is to ensure the authentication of the devices and the establishment of the shared symmetric keys required to secure the communications between the local device and the other devices of its community.

When the EAM of a device *a* detects a new device *b* that belongs to *a*'s community on the spontaneous network, it informs the Key Manager. If *a* does not already share a symmetric key with *b*, the Key Manager tries to authenticate *b* and to establish with it a shared key K*ab* by using the public key provided by the LPB. If the Key Manager already has a shared key K*ab* for, it uses it for mutual authentication with *b*. As we show later in this document, the symmetric keys established by the Key Managers are also used to secure the communications between the local device and the devices that belong to its community.

Usually, using point-to-point symmetric keys supposes to manage a very important amount of keys. In our case, the Key Manager manages and stores only the symmetric keys shared between the local device and each device it has to have a secure relation with. Consequently, the amount of keys managed by each device grows linearly with the number of devices of the community. This value is practically acceptable for the size of the typical devices communities we consider.

The *Message Filter* is the active part of our system. It enforces the security policy on the communications the local device takes part to, based on the configuration rules provided by the *Service Core*. It handles both incoming and outgoing communications. First, it ensures the security of the communications with the other devices of the community, and therefore creates a virtual private network (VPN) with the other devices of the community. It also controls the access of the devices that do not belong to the community to the locally provided services.

The Message Filter is itself made of two layers (cf. Fig. 4):

- The Cryptographic Layer, which, when necessary, checks the authenticity and decrypts the incoming messages, and encrypts and authenticates the outgoing messages.
- The Packet Filter Layer, which finely controls the access to the services provided by the local device.

Outgoing messages are first managed by the Packet Filter Layer that transmits them to the Cryptographic Layer. The later encrypts and authenticates them if required by the configuration rules, and sends the messages towards the network interface. Incoming messages are first managed by the Cryptographic layer that checks if they should be cryptographically protected. If they correctly are, or if cryptographic protection is not required, the messages are transmitted to the Packet Filter Layer. When the Packet Filter Layer receives a message from the Cryptographic Layer, it checks if this communication is in accordance with the rules describing the security policy provided by the

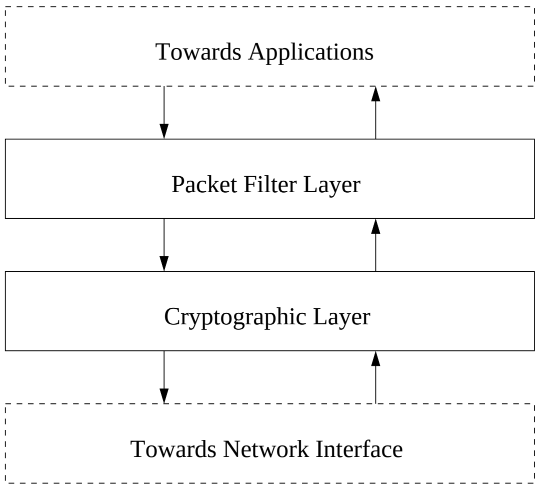

Fig. 4 – Architecture of the Message Filter.

Service Core, in which case it transfers it to the upper layers.

In addition to the dynamically generated configuration rules presented later in this document, the Cryptographic Layer and the Packet Filter Layer follows these given default rules. First, the incoming and outgoing messages are by default accepted unencrypted by the Cryptographic Layer. Second, by default, the incoming communications are rejected by the Packet Filter Layer, while the outgoing communications are accepted by it.

The Message Filter is also dynamically and automatically configured by the *Service Core* that generates the configuration rules for it using the information provided by the LPB, the EAM and the Key Manager. The configuration rules generation is triggered under two conditions. First, the LPB triggers it when the security policy is modified. Second, the EAM triggers it when the environment is modified, i.e. when a device that belongs to the community appears on the network or disappears from it, or when the IP address of such a device is modified.

The *Service Core* first generates the rules that deals with the communications that must be explicitly exchanged unencrypted. Indeed, some communications (and specially the communications used between the LPB, the EAM, and the Key Manager) must be exchanged without being encrypted, even between the devices that belong to the same community, since encrypting those communications either is impossible or would prevent the related services from working. Yet, those messages being exchanged between devices of the same community, the rules generated later by the Service Core would make the Cryptographic Layer to encrypt them automatically. For each such service, a rule is generated that states that the outgoing communication to these services have to be sent without being encrypted.

Then, the *Service Core* deals with the communications with the other devices of the community. On this topic, the abstract security policy states that the communications between devices of the community are authorized and must be secured. The Service Core knows the devices that belong to the community thanks to the LPB. It also knows the current IP address of the devices locally available in the spontaneous network thanks to the EAM. Finally, the Key Manager provides the symmetric keys that are currently shared with the devices of the community (if they have been established) and will be used to protect the communications. For each device *a* of the community for which the EAM has currently a valid address, the Service Core generates two rules dedicated to the Cryptographic Layer. The first one states that, in order to be accepted, the communications that supposedly come from *a* (i.e., that source address is the one provided by the EAM as being *a*'s) must be encrypted and authenticated using the key provided by the Key Manager. The second rule states that the outgoing communications to *a* have to be encrypted and authenticated using the key provided by the Key Manager. For each device *a* of the community for which the EAM has currently a valid address, the Service Core also generates a rule dedicated to the Packet Filter Layer that states that any incoming communication from *a* have to be accepted.

Finally, the *Service Core* generates the rules that deal with the public services. For each public service, it generates a rule dedicated to the Packet Filter Layer that states that any incoming communication for this service has to be accepted.

By dynamically and automatically configuring the Message Filter, the mechanism presented here takes into account the evolutions of the security policy and of the topology of the spontaneous network. At any time, the Message Filter is then properly configured to interact with the other available devices. It ensures the security of the communications with the devices of the community, and checks the legitimacy of the communications initiated by the devices that do not belong to the community but try to access to services offered by devices of the community.

7. CONCLUSION

In this article, we have proposed a fully decentralized service of automated configuration of the security services dedicated to communities of devices that communicate over spontaneous networks. Each device ensures itself the security of its interactions with the other devices. To that end, it manages its local knowledge of its community and of its security policy. Then, it uses a local message filter to enforce the security of the communications in which the local device takes part: communications with the devices that belong to the same community are encrypted and authenticated. Simultaneously, legitimacy of cross-boundary interactions is checked. This message filter is dynamically and automatically configured when the security policy and/or the environment is modified. Because this service does not rely on any infrastructure, it makes neither hypothesis on the network topology nor on the availability of a specific device. As such, it complies with the constraints induced by spontaneous networks.

We have developed a proof of concept of our approach over Linux. Each device has a 1024 bits RSA key pair, the provable identity being the SHA-1 hash value of the public key. IPsec is used to implement the Cryptographic Layer, while the Packet Filter Layer is based on NetFilter.

For our future works, we plan to focus on the expressiveness of the local security policy. The case of devices that, while not being in the community have privileged access to some services offered by some devices in the community should particularly be studied with more interest. We also think about studying the possibility of integrating mechanisms of automated management of short-term trust such as [2,8] in our approach.

8. REFERENCES

[1] Balfanz, D.; Smetters, D.; Stewart, P. & Wong, H. (2002), Talking to strangers: Authentication in ad hoc wireless networks, *in* 'Proceedings of the ISOC Network and Distributed Systems Security Symposium'.

[2] Cahill, V.; Gray, E.; Seigneur, J.; Jensen, C.; Chen, Y.; Shand, B.; Dimmock, N.; Twigg, A.; Bacon, J.; English, C.; Wagealla, W.; Terzis, S.; Nixon, P.; Segurendo, G.d.M.; Bryce, C.; Carbone, M.; Krukow, K. & Nielsen, M. (2003), 'Using Trust for Secure Collaboration in Uncertain Environment', *Pervasive Computing* 2(3).

[3] Capkun, S.; Hubaux, J.P. & Buttyan, L. (2003), Mobility helps Security in Ad Hoc Networks, *in* 'Proceedings of the Fourth International Symposium on Mobile Ad Hoc Networking and Computing'.

[4] Cheswick, W.R.; Bellovin, S.M. & Rubin, A.D., *Firewalls and Internet Security: Repelling the Wily Hacker*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. (2003)

[5] Corson, S. & Macker, J. (1999),'RFC 2501: Mobile Ad hoc Networking (MANET): Routing

Protocol Performance Issues and Evaluation Considerations'.

[6] Feeney, L.; Ahlgren, B. & Westerlund, A. (2001), 'Spontaneous networking: an application-oriented approach to ad hoc networking', *IEEE Communications Magazine*.

[7] Guttman, E., Autoconfiguration for IP Networking: Enabling Local Communication, *IEEE Internet Computing* 5(3), 81--86. (2001)

[8] Legrand, V.; Galice, S.; Ubéda, S. & Neuville, J. (2005), Identification pour les réseaux spontanés, *in* 'Actes de la quatrième conférence sur la Sécurité et Architectures Réseaux (SAR 2005)'.

[9] Montenegro, C. & Castelluccia, C. (2002), Statistically Unique and Cryptographically Verifiable (SUCV) identifiers and addresses, *in* 'NDSS'02'.

[10] O'Shea, G. & Roe, M. (2001), 'Child-proof authentication MIPv6 (CAM)', *ACM SIGCOMM Computer Communication Review* 31(2), 4--8.

[11] Perrig, A. & Song, D. (1999),Hash Visualization: a New Technique to improve Real-World Security, *in* 'International Workshop on Cryptographic Techniques and E-Commerce (CrypTEC '99)', pp. 131--138.

[12] Prigent, N.; Bidan, C.; Andreaux, J. & Heen, O. (2003),Secure Long Term Communities in Ad Hoc Networks, *in* 'Proceedings of the First ACM Workshop on Security in Ad Hoc and Sensor Networks (SASN), held in association with the Tenth ACM Conference on Computer and Communication Security (CCS)'.

[13] Prigent, N.; Bidan, C.; Heen, O. & Courtay, O. (2005), Configuration automatisée des services de sécurité de communautés d'appareils dans les réseaux spontanés, *in* 'Actes de la quatrième conférence sur la Sécurité des Architectures Réseaux (SAR 2005)'.

[14] Stajano, F. (2001), 'The Resurrecting Duckling -- What Next?', *Lecture Notes in Computer Science* 2133, 204--211.

Christophe Bidan *obtained his PhD in Computer Science from the Université de Rennes 1, France, in May 1998, in the information systems security field. After his PhD, he spent a one year postdoc at the Imperial College of London. Then, he has worked as security expert for Gemplus from 1999 until 2000. Since 2000, he is a research assistant at Supélec Rennes, France, in the Network and Information Systems Security Group, where he investigates security solutions in next generation networks (wireless and ad hoc networks, home networks).*

Nicolas Prigent *is a research engineer in the Security Laboratory of Thomson R&D Rennes, France, where he prepares a PhD dealing with the security of dynamic and self-configurable networks. He holds a M. Sc. in Computer Science from the University of Rennes 1. His research interests include computer science, networks, and secure systems.*