

A solution to the problem of secure boundary in home ad hoc networks*

Nicolas Prigent^{†‡} Jean-Pierre Andreaux[†] Christophe Bidan[‡] Olivier Heen[†]

October 8, 2003

Abstract

Home networks are nowadays an application of choice for ad hoc networks technology. A home network is made of a set of devices, linked by a long term relation, that interact seamlessly to offer enhanced services to users. This set evolves when a user buys, sells, or loses devices. In order to ensure the security of home networks, it is first necessary to define which devices belong to a given network, and are consequently in the security perimeter. In this paper, we propose a user-friendly distributed mechanism to define which devices belong to a given home network. This mechanism takes into account the various possible evolution operation of the home network.

Keywords: ad hoc networks, home network, security.

1 Introduction

A home ad hoc network [3] consists in a group of devices that interact seamlessly to offer enhanced services to users without requiring any dedicated server or extensive configuration.

By opposition to infrastructure-based corporate networks, home ad hoc networks¹ have properties [1, 4, 8] that impact on the use of classical security solutions. First, there is no guarantee of device connectivity: no device can be assumed to be always present, and there can be no centralized point

of control. Second, the boundaries of home ad hoc networks are poorly defined, by opposition to the one of wired LANs that can be enforced using the infrastructure. Third, resources and skills in security administration are scarce: users neither have time nor skills to manage security. Worse, they are often considered as the weakest link of security [9].

Nevertheless, home networks will not be broadly deployed if their security is not ensured.

In order to secure home networks, the first step is to define securely 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.

In home networks, communicating devices (e.g. computers, TV sets, set-top boxes, PDAs, printers, etc.) are linked by long term relations. While the topology could evolve quite frequently like in any classical ad hoc network, and even if some devices could be temporarily unreachable, the devices composing the home network will not vary very often: a device enters the network *a priori* for a 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.

This paper present the concept of secure long term community as a way to mark the boundary between the devices belonging to a given home network and the others. More particularly, we propose a mean to set up and manage a long term virtual private network between devices placed under a common policy.

*This paper is a short version of *Secure Long Term Communities in Ad Hoc Networks*, accepted to ACM SASN 2003

[†]Thomson Multimedia R&D France,
Nicolas.Prigent@thomson.net, Jean-Pierre.Andreaux@thomson.net, Olivier.Heen@thomson.net

[‡]Supelec, Rennes, France, Christophe.Bidan@supelec.fr

¹Starting from here, we will use “SOHO and home networks” for short, implying “SOHO and home ad hoc networks”.

2 Secure Long Term Communities

Albers et al. [1] define a community as “a set of devices related by a trust relation”. The devices belonging to the same home network can reasonably trust each others, and consequently form a community. Because the trust relation between the devices lasts *a priori* for a long time, this is a long term community.

All the devices in a home network do not always belong to one and the same person. For example, 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 [5] that states a community as “a group linked by a common policy”, we consider the case of possibly multiple authorities sharing the same policy. By opposition to [2], 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.

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

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

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 devices: an attacker must not be able to eavesdrop messages.

- Authenticity of communications between devices: an attacker must not be able to modify, to inject or to replay messages.

2.2 Robustness regarding physical constraints

Like in classical ad hoc networks, devices connectivity cannot be guaranteed in 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 Λ be a community and a a device, $a \in \Lambda$. We call $\Lambda(a)$ the devices belonging to Λ that a can authenticate as so, i.e. the knowledge a has of Λ .
- We call $\Phi(a)$ the devices that a can communicate bi-directionally with at a given time. All the devices of $\Phi(a)$ do not necessarily belong to Λ , $a \in \Lambda$.
- 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 a given time, secure long term communities should offer the following properties.

First of all, any two devices ‘ a ’ and ‘ b ’ of the same community Λ , $a \in \Lambda(b)$ and $b \in \Lambda(a)$ 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.

Moreover, a community should be able to evolve without requiring a specific device: any two devices should be able to get into the same community while not able to reach any other device from their own community. For example,

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.

Finally, 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.

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 Λ be the community, and a the first element of this community. The initialization is defined by:

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

2.3.2 Insertion and merge

A community can accept new devices. After a new device 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 Λ , we have:

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

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 Λ , and b another device belonging to the community Λ' . The community resulting from the merge of the two communities Λ and Λ' contains the devices of both these communities.

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

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 Λ , we have:

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

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 would use it to access the rest of the community. Banishment consists in removing a member from the 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 Λ , we have:

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

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 knowledge of it, we would reach the global consistency.

However, because some devices may be unreachable during the evolution of the community, it may not be possible to ensure global consistency. 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 respective knowledge of the community must get consistent.

Loose consistency consists in the fact that, a being a device in the community Λ , all the devices of $\Delta(a)$ share the same knowledge $\Lambda(a)$

of Λ , and $\Lambda(a)$ is the most up-to-date local knowledge of Λ in $\Delta(a)$. If all the devices of a community communicate altogether again, we reach global consistency of the community.

3 A New Approach For Securing Long Term Communities

In the mechanism we propose, each device manages its local knowledge of the community. The user initiate changes by informing locally some of the devices of modifications of the community, and the devices share information to keep their knowl-

edge consistent with the reality. There is no central information nor central element: each device considers itself as the central element, around which the whole community evolves.

3.1 Provable Identity

First, devices must be able to identify and authenticate the other devices of their community, and to communicate securely with them. To do so, each device has its own provable identity, which is an identity that anyone can check, but which is very hard to impersonate. For instance, the public key of a public/private key pair is a provable identity: a device pretending to be 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.

Using their respective provable identities, two devices can create a point to point secure channel by encrypting their messages with the public key 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.

SUCV [6] and CAM [7] are other examples of mechanisms of provable identity.

3.2 Local Knowledge

In our proposal, each device a securely manages its local knowledge $\Lambda(a)$ of its community Λ , using the provable identities of the other members of its community.

a will communicate freely with another device b only if it know that b belongs to its community, i.e. $b \in \Lambda(a)$. In result, two device will communicate freely only if each one knows that the other one belongs to its own community.

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

- Mutual trust
- Unilateral trust
- Distrust

When a knows b as mutually trusted, it means that b belongs to Λ and has already been met by

a : a trusts b , b trusts a , and a knows that b trusts a . a also has a certificate signed with b 's provable identity that proves that b considers a as being in its community.

When a knows b as unilaterally trusted, it means that b belongs to Λ , a trusts b , but a never met b : locally, a does not know if b knows that a belongs to the community, and consequently if it trusts a . Every b that a knows as unilaterally trusted has been introduced to it by a device c , that is mutually trusted by a . b can be either mutually trusted or unilaterally trusted by c . For each such b , a has a chain of certificates provided by c that proves that b trusts a . When a will meet b , it will be able to provide this chain of certificates to prove to b that it should trust a .

When a knows b as distrusted, it means that b , while being locally known, does not belong to $\Lambda(a)$ anymore. b was formerly in Λ , but has been banished or removed. a will not accept any proof from b that they belong to the same community. This is equivalent to the issue of revocation for a certificate.

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 $MT(a)$ and $UT(a)$ being respectively the sets of devices a has a mutual trust relation with, or a unilateral trust relation with, we have:

$$\Lambda(a) = MT(a) \cup UT(a)$$

$DT(a)$ is the set of devices that a knows as not being in the community anymore, i.e., that have been explicitly removed or banished. By construction, $MT(a)$, $UT(a)$ and $DT(a)$ are totally disjoint.

3.3 Secure evolution of the community

a being a device in Λ , any modification of $\Lambda(a)$ may not be due to a real evolution of Λ : $\Lambda(a)$ can evolve because a device $c \in \Lambda(a)$ informed a that Λ has been modified. In this case, the modification of $\Lambda(a)$ is just a synchronization (described in §3.4) of a 's local view with c 's one.

Λ evolves only when $\Lambda(a)$ evolves and a is the first device in Λ where the modification occurs.

An evolution of the community is initiated on any device a belonging to it by the local authority (i.e. the user): she or he informs the local device a that a device has to be inserted, removed or 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 §3.4).

Requests for evolution being security-relevant, a has to authenticate the authority, i.e. the user. This authentication is strictly local to a : a can use the best suited authentication mechanism, that is not necessary the same as the one used on other devices of the community. Moreover, because a user does not have a unique representation in the community, there is no such information around which the whole community would be organized, and our proposal is really decentralized.

3.3.1 Initialization

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

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

a 's view of Λ 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.

3.3.2 Insertion

As seen in §3.2, 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 symmetric².

When a inserts b in its community, b also inserts a . Because they both know they are being inserted by the other one, each of them inserts the other as a mutually trusted device:

$$\begin{aligned}\text{MT}(a) &:= \text{MT}(a) \cup \{b\} \\ \text{MT}(b) &:= \text{MT}(b) \cup \{a\}\end{aligned}$$

Moreover, a (resp. b) issues a certificate to b (resp. a) that proves that it considers b (resp. a) as being in its community.

Inserting manually the provable identity of a device in the other is not a user-friendly approach. A way to solve this problem would be to use a secure side channel to transmit the provable identities as described in [2, 10, 11].

²We 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 [10].

3.3.3 Removal

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

$$\begin{aligned}a \in \Delta(b), \text{MT}(a) &:= \text{MT}(a) \setminus \{b\}, \\ \text{UT}(a) &:= \text{UT}(a) \setminus \{b\}, \\ \text{DT}(a) &:= \text{DT}(a) \cup \{b\}\end{aligned}$$

b then clears its local information:

$$\begin{aligned}\text{DT}(b) &:= \text{DT}(b) \cup \text{UT}(b) \cup \text{MT}(b) \setminus \{b\} \\ \text{MT}(b) &:= \{b\}, \text{UT}(b) := \emptyset\end{aligned}$$

and consequently, we have:

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

If no device is available during b 's removal, ($\Delta(b) = \emptyset$ at the time of the removal), the user could, for more security, inform one of the devices still belonging to the community that b does not belong to it anymore, using the thereafter explained banishment mechanism.

3.3.4 Banishment

The banishment of a device b from the community Λ 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 $\text{MT}(a)$ or $\text{UT}(a)$ and inserting it in $\text{DT}(a)$:

$$\begin{aligned}\text{MT}(a) &:= \text{MT}(a) \setminus \{b\}, \\ \text{UT}(a) &:= \text{UT}(a) \setminus \{b\}, \\ \text{DT}(a) &:= \text{DT}(a) \cup \{b\}\end{aligned}$$

Consequently, we have:

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

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

3.4 Distributed loose consistency

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

However, to ensure the loose consistency in the community, each device shares its local knowledge and synchronize its information with the others when they are reachable. A device trusts the other devices in its community to provide information about insertions, removals or banishments that occurred in the community. Thus, the trust relation is transitive: a , b and c being three devices, if c

trusts b as being in its community Λ , and b trusts a as being in Λ , then c trusts a as being in Λ .

This transitive property is valid in our proposal, because all the devices in the same long term community are submitted to the same policy and trust each others. When a device a inserts a device 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 devices in 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 (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 distrusted³. 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 local knowledge.

3.4.1 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.

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.

Finally, they exchange the devices they know as unilaterally trusted. Then, a and b exchange all the certificates that a (resp. b) previously received from the devices belonging to $MT(a)$ (resp. $MT(b)$), and that do not carry on distrusted devices.

Thanks to the transitive property of the trust relation, when a (resp. b) will meet a device c with which it has an unilateral trust, it can provide to c all the certificates proving that c should trust a (resp. b). For example, if d previously met c , d

³If 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.

has issued to c a certificate proving that it trusts c . When c meets b , d is included in $UT(b)$, and b receives the certificates that prove that c trusts b and that d trusts c . When b meets a , c and d are included in $UT(a)$, and a receives the certificates to prove that b trusts a , c trusts b and d trusts c . Then, when a meets c or d , it is able to provide the certificates informing c or d that they should trust a .

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

3.4.2 Evolution of Φ

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

If $b \in MT(a)$ (case 1), $a \in MT(b)$: 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 local views $\Lambda(a)$ and $\Lambda(b)$ may have evolved. Consequently, a and b may have to synchronize their views (see §3.4.1).

If $b \in UT(a)$ (case 2), a first has to provide to b the certificates proving to b that it is trustable. b checks these tickets, both a and b mark each other locally as mutually trusted, and they exchange the certificates proving they trust one another. Using our notation, Now that $a \in MT(b)$ and $b \in MT(a)$, they can synchronize their local views of the community as described in §3.4.1.

In the case where the chain of certificates provided by a device a involves a certificate issued by a device once in the community but now distrusted, 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 simply unknown by the other members of the community. So, the user 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 views of the community, f will

inform a that e is now distrusted, and a 's view of the community will consequently be up to date.

3.4.3 Evolution of Λ

$\Delta(a)$ can also evolve when a new device b enters $\Lambda(a)$. We already detailed the insertion phase in §3.3.2 and the synchronization between the devices a and b (see §3.4.1). 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.

For all $c \in \Delta(a)$ (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 §3.4.2.

$\Lambda(a)$ can also evolve because a has marked a device in $DT(a)$. In this case, it synchronizes with the devices in $\Delta(a)$, as defined in 3.4.1.

4 Conclusion

We have presented a new user-friendly distributed approach to set up and maintain a secure long term community in a home ad hoc network.

Our proposal ensures the required security services for the members of the community (even if they communicate over insecure channels), 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. 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 networks, that require to build long term communities of devices while requiring light user involvement in security mechanisms. It would now be interesting to enable automatic security policy management in the community, taking into account the possibly conflicting interests between the authorities. We also plan to study how a device not belonging to the community could be temporarily integrated into it.

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] 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.
- [3] S. Corson and J. Macker. RFC 2501 : Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations, Jan. 1999.
- [4] L. Feeney, B. Ahlgren, and A. Westerlund. Spontaneous networking: an application-oriented approach to ad hoc networking. *IEEE Communications Magazine*, June 2001.
- [5] Merriam-Webster. Merriam-webster online dictionary, www.webster.com.
- [6] C. Montenegro and C. Castelluccia. Statistically Unique and Cryptographically Verifiable (SUCV) identifiers and addresses. In *NDSS'02*, Feb. 2002.
- [7] G. O'Shea and M. Roe. Child-proof authentication for mipv6 (cam). *ACM SIGCOMM Computer Communication Review*, 31(2):4–8, 2001.
- [8] N. Prigent, C. Bidan, O. Heen, and A. Durand. Sécurité des réseaux domestiques : optimaux les grands remèdes ? In *Symposium sur la Sécurité des Technologie de l'Information et des Communications*, pages 41–52, 2003.
- [9] S. W. Smith. Humans in the loop: Human-computer interaction and security. *IEEE Security & Privacy*, June 2003.
- [10] F. Stajano. The Resurrecting Duckling – What Next? *Lecture Notes in Computer Science*, 2133:204–211, 2001.
- [11] 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.