Thèse N^o : 28-2008

UNIVERSITÉ DE LIMOGES

ECOLE DOCTORALE Science - Technologie - Santé

Faculté de Sciences & Techniques

Laboratoires : XLIM & LaBRI

THÈSE

pour obtenir le grade de

DOCTEUR

SPÉCIALITÉ : INFORMATIQUE

présentée et soutenue le 4 septembre 2008 par

Eve Atallah

Une solution pour l'établissement non planifié de groupes sécurisés permettant des communications sûres dans les réseaux MANets purs

---

Thèse dirigée par
Jean-Pierre BOREL (XLIM) et Serge CHAUMETTE (LaBRI) et suivie par
Pierre-François BONNEFOI (XLIM)

Rapporteurs

Christian LAFOREST Maître de Conférences HDR, IBISC, Université d'Evry

Jury

Rapporteurs

Congduc PHAM ..... Professeur (LIUPPA, Université de Pau)

Examinateurs

Jean-Pierre BOREL ..... Professeur (XLIM, Université de Limoges)
Serge CHAUMETTE ..... Professeur (LaBRI, Université Bordeaux 1)
Jean-Louis LANET ..... Professeur (XLIM, Université de Limoges)
Pierre-François BONNEFOI Maître de Conférences (XLIM, Université de Limoges)
Olivier HEEN ..... Spécialiste industriel (INRIA, Thomson R&D)
Philippe ROOSE ..... Maître de Conférences (LIUPPA, Université de Pau)

– 2008 –

Remerciements

Cette thèse a été financée par la région Limousin. Les travaux ont été réalisés dans le cadre d'une collaboration entre les laboratoires de recherche informatique de Limoges (XLIM) et de Bordeaux (LaBRI).

Je tiens tout d'abord à remercier mes directeurs de thèse et encadrants Jean-Pierre Borel, Serge Chaumette et Pierre-François Bonnefoi pour m'avoir permis de travailler sur un sujet de thèse aussi captivant. Merci en particulier à Jean-Pierre Borel et Pierre-François Bonnefoi pour m'avoir accueillie et encadrée au sein du laboratoire de Limoges (XLIM) et à Serge Chaumette (LaBRI) pour m'avoir intégrée dans son équipe dès le DEA et pour m'avoir encadrée et guidée tout au long de cette thèse.

J'exprime aussi mes remerciements à Jean-Louis Lanet pour m'avoir fait l'honneur de présider le jury de soutenance. Je remercie aussi très sincèrement Christian Laforest, Congduc Pham et Philippe Roose, rapporteurs de cette thèse, pour le temps et l'attention qu'ils ont accordés à ces travaux ainsi que pour leurs questions et leurs suggestions qui m'ont permis d'enrichir ce document. Je tiens de même à remercier Olivier Heen pour l'intérêt qu'il a porté à ce travail en acceptant de participer au jury et pour ses remarques et ses conseils avisés.

Je voudrais également remercier Damien Sauveron pour l'aide qu'il m'a apporté en prenant toujours le temps de répondre à mes questions.

Je remercie bien sûr les habitants de la CVT pour l'excellente ambiance et l'entre-aide qui y règnent. Un grand merci notamment à Arnaud, Lionel, Jérémie, Jonathan, les Rémi, Fabien et Julien.

Je remercie aussi chaleureusement mes amis et tout particulièrement Aurélie pour être toujours là ainsi que Marina et Emilie pour leur précieux soutien moral.

Je remercie ma famille et tout particulièrement ma mère pour son soutien inconditionnel, mon père pour m'avoir toujours encouragé dans mes études, Zazy pour son sens de la dé-dramatisation ainsi que Marie-Thérèse, François, Cécile et Xavier pour avoir fait ce long déplacement afin de participer à ma soutenance. Je remercie aussi tout ceux qui m'ont tant apporté dont Caroline toujours dans mon cœur, Daniel, Titi et bien sûr ma petite Elsa. Je remercie enfin Armand pour sa présence à mes côtés et pour sa patience.

Table des matières

Table des figures

Introduction

Les réseaux informatiques, déjà très présents dans la vie quotidienne, en particulier depuis l'apparition d'Internet dans les foyers, sont aujourd'hui devenus ubiquitaires, c'est-à-dire accessibles partout, le réseau étant omniprésent et les équipements se déplaçant avec leurs utilisateurs. Une multitude d'usages apparaît. Il est donc crucial de proposer des mécanismes permettant d'utiliser ces réseaux de manière sécurisée. C'est le sujet de cette thèse.

Contexte de travail

Les réseaux informatiques locaux encore appelés intranets sont des réseaux statiques, fixes, et donc très contraignants. Ils sont un moyen de connecter les individus d'une même organisation. D'importantes avancées pour apporter plus de souplesse concernant le déploiement matériel des réseaux et plus de liberté de déplacement aux utilisateurs ont été réalisées. Tout d'abord, l'utilisation d'Internet a apporté la mobilité en permettant aux usagers de se connecter depuis l'extérieur de la zone géographique de leur intranet. Ensuite, les réseaux ad hoc ont apporté la spontanéité de constitution d'un réseau physique (n'importe où, n'importe quand) grâce à un système d'échanges entièrement pris en charge par les dispositifs utilisateurs eux-même et ne nécessitant donc aucune infrastructure. Les MANets (*Mobile Ad hoc Networks*) apportent la dimension de mobilité : ce sont des réseaux ad hoc dont les dispositifs, étroitement liés à leurs propriétaires, se déplacent avec eux. On parle de MANets purs lorsque les applications qui s'y exécutent n'utilisent ni centralisation, ni administration (on appelle administration une entité chargée de la gestion du réseau).

Dans la suite, nous entendons par réseau l'ensemble des éléments permettant la communication entre les dispositifs qui le composent. L'utilisation d'un réseau par les utilisateurs se fait dans le cadre de groupes, un groupe étant un ensemble d'utilisateurs partageant le même intérêt.

Existant

On sait aujourd'hui établir des réseaux n'importe où, n'importe quand. En réalité, si c'est le cas d'un point de vue matériel, les difficultés rencontrées pour rendre ces solutions réellement utilisables ont jusqu'à présent limité leur emploi à des situations très spécifiques.

Dans la plupart des solutions développées, les MANets puisque c'est d'eux dont il s'agit,

sont traités comme des réseaux classiques. Leur utilisation repose alors sur des hypothèses telles que la prise en charge de certaines opérations par une administration externe, une faible mobilité de certains nœuds pour pouvoir centraliser des données, etc ; pour garantir la sécurité on suppose en général, et c'est un autre exemple de contrainte, une proximité géographique des nœuds. Ainsi, si la spontanéité matérielle des MANets est bien exploitée, leur utilisation est, cependant, toujours assujettie à une phase de préparation par une organisation globale. Or, pour que la constitution d'un groupe et le fait d'y participer soient spontanées, ce qui est le but à atteindre, il faut que les échanges de données puissent être décidés dynamiquement, sans planification.

But de la thèse

Dans ce travail, nous mettons en place un service de sécurisation des communications dans les réseaux MANets purs (sans centralisation ni administration), ce qui devrait permettre de réellement les exploiter. Nous souhaitons faire des MANets un outil utilisable au quotidien, spontané mais offrant des garanties de sécurité.

Pour cela, il faut ajouter aux caractéristiques des MANets « n'importe où, n'importe quand », la notion de « entre n'importe qui et n'importe qui », c'est-à-dire indépendemment de tout rapport humain préexistant : les individus participant doivent pouvoir être inconnus les uns des autres et donc ne pas avoir pu planifier leurs échanges à l'avance.

Notre travail vise donc à offrir la spontanéité de création de groupes afin de ne plus se limiter à des groupes prédéfinis (pas de préparation préalable entre les membres).

Un exemple d'utilisation serait la formation d'un groupe entre des membres de deux laboratoires différents lors d'une conférence (fig. 1). Les membres d'un même laboratoire (en rouge ou en bleu sur la figure) pourraient former un groupe car ils partageraient un secret commun distribué avant la conférence ($S1$ ou $S2$). En revanche, il serait difficile pour les membres de deux organisations différentes, n'ayant pas de secret commun, d'organiser un groupe fermé (i.e. restreint à un ensemble fini d'individus choisis pour leurs connaissances, capacités, etc.). Nous proposons une solution pour cela : deux individus, chacun d'un laboratoire différent décident de créer un groupe. Ils attribuent chacun à ce groupe leur propre moyen d'authentification ($S1$ et $S2$). Pour y entrer il suffit de connaître l'un des deux uniquement. Ainsi, chaque membre d'un des deux laboratoires peut participer au groupe commun sans prendre connaissance du secret de l'autre laboratoire. Notre solution assure aussi la publicité de ce nouveau groupe auprès des utilisateurs présents.

Problématiques

Nous introduisons ci-dessous trois des problèmes fondamentaux que nous traitons dans cette thèse.

Gestion des identités. Pour permettre la protection des échanges, la différenciation de chacun des dispositifs/utilisateurs est nécessaire. En effet, l'identification unique des utilisateurs permet d'apporter la sécurité des échanges privés (ciblés) ainsi que la traçabilité,

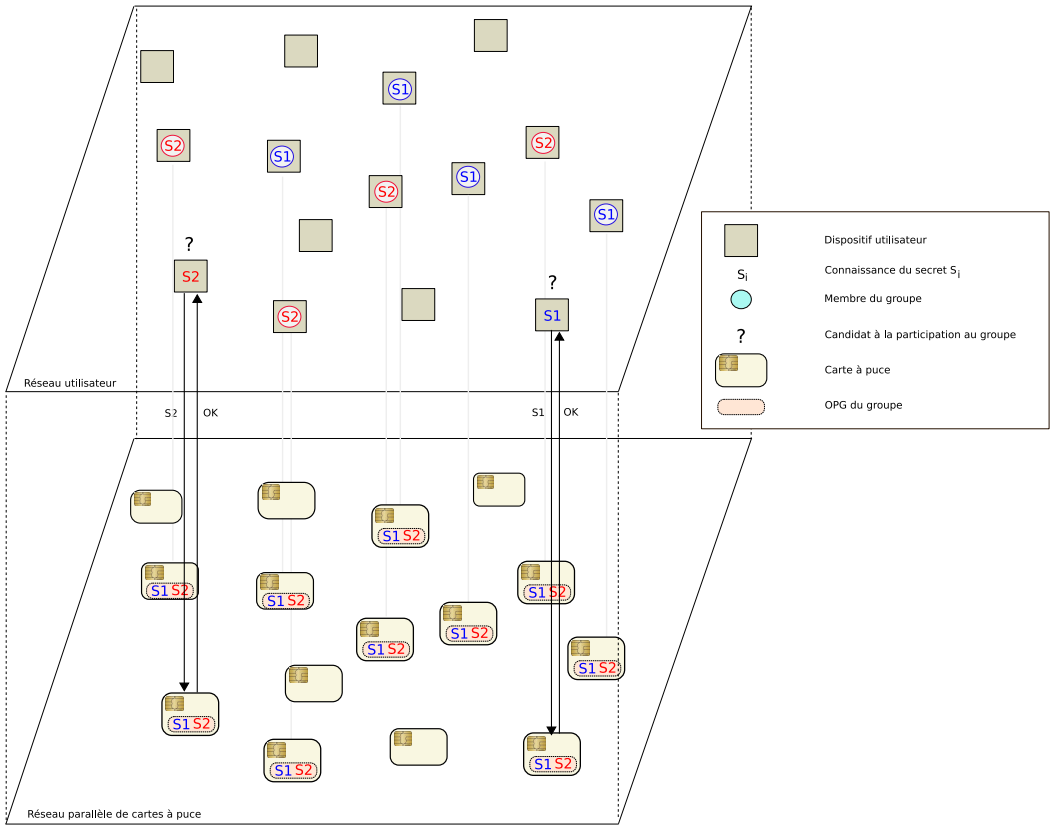


FIG. 1 – Exemple de constitution de groupes

caractéristique nécessaire à la prise en compte des actes malveillants et à leur imputabilité. Les identités doivent être construites de manière décentralisée ce qui pose aussi le problème de leur diffusion.

Notion de groupe et d'intérêt. Ce qui définit le groupe, c'est son but, c'est-à-dire l'intérêt de ses membres à en faire partie. Les solutions existantes proposent soit une entente préalable des membres réalisée hors du réseau, soit l'exploitation de canaux naturellement sûrs (liaison infrarouge par exemple) pour échanger les informations nécessaires à l'initialisation du groupe et à la gestion (entrées, sorties, communications, etc.) de ses membres. Or, pour que la constitution et la participation à un groupe soient spontanées il faut que les échanges de données puissent se faire par le réseau, et non à l'avance.

Gestion des accès au groupe. La problématique provient du caractère dynamique que l'on souhaite donner à la gestion des accès (arrivées et départs d'éléments d'un groupe). Les éventuels candidats à l'intégration doivent être informés de l'existence et des caractéristiques de tout nouveau groupe. La procédure de contrôle d'accès (vérification que le candidat a la capacité et le droit de participer ou contribuer au but) doit pouvoir être réalisée auprès de n'importe quel nœud. Dans les solutions existantes ceci est souvent géré par un nœud spécifique, centralisateur ; des informations préexistantes au réseau lui sont

fournies pour qu'il puisse faire ces contrôles. Ceci est incompatible avec nos objectifs (ni centralisation, ni administration, ni planification).

Notre approche

Nous souhaitons développer une solution de sécurité dans un contexte où on ne compte sur aucune centralisation ou administration pendant la vie des groupes, ni même sur aucune organisation humaine planifiant ces groupes. Ceci engendre deux problématiques principales que nous devons considérer : tout d'abord, si les utilisateurs sont libérés des points centraux et des structures administratives, ils se retrouvent en contre partie seuls responsables du bon fonctionnement du réseau. Les opérations de contrôle relatives à la sécurité sont alors à leur charge alors qu'aucun d'entre eux ne peut être considéré comme forcément honnête a priori. Le problème est donc de permettre la réalisation d'opérations de contrôle de sécurité par les dispositifs utilisateurs eux-mêmes sans qu'un nœud malveillant puisse causer des dommages.

Par ailleurs, les organisations humaines créent des groupes dans un but particulier. Au sein de ce groupe, les communications ont donc une raison d'être. Si la notion d'organisation humaine est éliminée, alors les utilisateurs d'un réseau n'ont plus a priori de raison de communiquer. Il est donc nécessaire de fournir un mécanisme permettant aux utilisateurs de se rassembler autour de buts communs en utilisant uniquement les canaux de communications du réseau.

Dans notre travail, nous englobons la totalité de ces problématiques autour de deux points : la gestion des identités pour pouvoir (entre autres) cibler les communication et établir des frontières ; la gestion de la formation des groupes pour cadrer les communications autour d'un but commun (sans but partagé les communications n'ont pas de raison d'être).

Identités. Nous souhaitons que l'identité d'un nœud soit unique (aucun autre nœud ne doit avoir la même) et non-répudiable, et que l'appartenance à un groupe ne soit possible que sous une seule identité. De par la nature des MANets purs (aucune centralisation, ni administration), la construction des identités doit se faire de manière décentralisée ce qui pose aussi le problème de leur diffusion.

Groupes. Nous souhaitons que la création et la formation d'un groupe soient entièrement spontanées et ne nécessitent aucune structure supplémentaire. La diffusion des caractéristiques d'un groupe (dont son but) ainsi que les mécanismes permettant une gestion dynamique des membres demandent aussi à être réalisés de manière décentralisée. Les éventuels candidats à l'intégration doivent être informés de l'existence et des caractéristiques de tout nouveau groupe. La procédure de contrôle d'accès (vérification que le candidat a la capacité et le droit de participer ou de contribuer au but) doit pouvoir être réalisée auprès de n'importe quel nœud. Dans les solutions existantes ceci est souvent géré par un nœud spécifique, centralisateur ; des informations préexistantes au réseau lui sont fournies pour qu'il puisse faire ses contrôles. Ceci est incompatible avec nos objectifs.

Notre solution

Principes

Ce sont des cartes à puce, une par nœud, qui assurent un contrôle strict des données qu'elles contiennent et des traitements qu'elles réalisent (respect des règles de sécurité, des protocoles, etc.). Elles protègent toute donnée ou application qu'elles hébergent d'une action non autorisée (modification, injection, lecture, etc.). Néanmoins, sorties de la carte, les données ne sont plus protégées. Nous proposons donc qu'un canal d'échange sécurisé entre les cartes soit créé par l'ajout en usine d'éléments cryptographiques. L'ensemble de ces canaux d'échange constitue un réseau parallèle innaccessible aux utilisateurs. Pour créer ces canaux, l'utilisation d'une clé globale à toutes les cartes est la solution retenue. Cette clé ne sera pas utilisée dans toutes les phases de sécurisation, mais permettra par exemple de générer d'autres clés et de vérifier des certificats. A la sortie d'usine, les cartes deviennent interchangeables entre les utilisateurs. L'autorité générant cette clé globale n'a donc aucune connaissance des groupes futurs ni aucune information qui lui permettrait de favoriser une organisation humaine particulière. Ce réseau de cartes permet de sécuriser les échanges entre les utilisateurs de manière indépendante de toute intention préexistante.

Organisation de la suite de ce document

Ce document est découpé en cinq chapitres.

Le premier chapitre consiste en une présentation des réseaux et des notions de sécurité associées. Nous nous intéressons tout particulièrement à ce qui influe sur le développement de services de sécurisation et plus particulièrement dans le domaine de la sécurité des communications.

Dans le deuxième chapitre, nous nous intéressons aux réseaux de type MANet. Nous en faisons une présentation générale et nous insistons sur leurs spécificités. Nous en déduisons les problématiques engendrées au niveau de la sécurité des communications et nous précisons nos objectifs et les mécanismes à mettre en œuvre qui en découlent (nous parlons de besoins).

Dans le troisième chapitre, nous mettons en évidence l'atout que représentent les cartes à puce pour la mise en place de notre solution de sécurité en nous appuyant sur l'analyse des besoins précédemment établis. Nous présentons les cartes et détaillons notre approche en expliquant leur rôle et leur intérêt à chaque étape de notre solution.

Dans le quatrième chapitre nous présentons un algorithme distribué d'élection de *leader*. Le but est d'utiliser les différents modules de sécurité de notre solution afin d'en tester la facilité d'intégration par une autre application et de vérifier leurs apports du point de vue de la sécurité.

Le chapitre cinq présente la partie technique de notre travail à savoir l'application SManet. Nous y décrivons les outils de développement utilisés, les détails de l'implémentation

réalisée et les tests effectués.

Enfin, dans le dernier chapitre nous décrivons le fonctionnement de l'application SManet ainsi que des exemples de scénarios qui mettront en évidence le besoin et l'intérêt de cette application et les besoins auxquels elle répond.

Nous concluons en présentant un bilan de nos contributions et des directions de recherche futures qui en découlent.

Chapitre 1

Introduction aux réseaux et à la sécurité informatique

Sommaire

Dans ce premier chapitre, nous présentons les réseaux et un certain nombre de notions qui leur sont rattachées, notamment celles dont dépend le développement d'applications ou de services liés à la sécurité. La première section traite des réseaux en général et la deuxième concerne plus particulièrement les MANets. Nous proposons ensuite une introduction aux problématiques liées à la sécurité des communications. Pour terminer, nous étudions, en nous appuyant sur ce qui a été présenté dans les sections précédentes, les problématiques et les solutions qui existent pour gérer la sécurité des communications dans un réseau.

1.1 Présentation générale des réseaux

Dans un premier temps nous présentons les différents types de réseaux existants et nous mettons en évidence la diversité de leurs caractéristiques. Nous proposons ensuite un ensemble de définitions et de classifications dont nous nous servirons dans le reste de ce document. Enfin, nous décrivons en détail les caractéristiques principales des réseaux et étudions l'influence de ces caractéristiques sur le développement d'applications.

1.1.1 Les différents types de réseaux existants

Nous étudions ici les différents réseaux les plus classiques, ce qui nous permet de constater la diversité de leurs caractéristiques.

a) Intranets classiques

Les intranets sont des réseaux locaux utilisés par les membres d'un même groupe humain organisé (un réseau d'entreprise par exemple). Ils reposent sur une infrastructure matérielle de routage contraignante qui doit être installée et configurée par des administrateurs. Les applications y sont de type client/serveur, les serveurs étant le plus souvent des dispositifs fixes reliés au réseau par une technologie filaire. Les postes clients peuvent être fixes ou disposer d'une mobilité (limitée) en cas d'utilisation de routeur sans fil.
Dans ces réseaux, toute la sécurité repose sur la centralisation des données et un contrôle assuré par des administrateurs.

b) Réseaux nomades

Dans un réseau nomade, l'infrastructure principale est du même type que celle des réseaux intranets. En revanche, les utilisateurs peuvent se déplacer physiquement mais tout en gardant un contact avec cette infrastructure. L'extension de l'accès à l'intranet depuis l'extérieur de celui-ci peut se faire au moyen d'une technologie non-filaire (GPRS [33]) ou filaire (utilisation du maillage Internet avec une technologie de sécurité de type VPN [46]). Les utilisateurs bénéficient ainsi d'une plus grande liberté dans leurs déplacements grâce à une « mobilité des accès » (cf. sous section (e)).
Un exemple d'utilisation classique est celui d'un employé (en mission à l'extérieur) consultant une base de données de l'intranet de son entreprise via une liaison Internet.
Dans ces réseaux, en plus des problématiques des intranets classiques, s'ajoute celle de la protection des accès depuis l'extérieur.

c) Réseaux domestiques

Les réseaux domestiques sont des intranets de petite taille ayant pour particularité d'exister dans l'environnement privé des utilisateurs. Tous les dispositifs du réseau se trouvent alors dans une limite géographique très restreinte et sont faciles à contrôler et à sécuriser. En théorie, aucune attaque de l'intérieur n'est à craindre. En revanche, certaines peuvent provenir de l'extérieur, notamment si des technologies non filaires sont utilisées. En effet, les ondes traversant les murs, un voisin pourrait tenter de prendre le contrôle de dispositifs ne lui appartenant pas.

d) Réseaux peer-to-peer/ad hoc

Les réseaux peer-to-peer/ad hoc diffèrent des réseaux précédents par l'absence de serveurs. Toutes les applications reposent uniquement sur les machines utilisateurs pour réaliser leur tâche. La contribution de chacun devrait donc être d'égale importance. Ces réseaux permettent une installation simplifiée et rapide puisqu'aucune architecture supplémentaire (autre que les dispositifs utilisateurs) n'est requise. En revanche, la sécurité est beaucoup plus difficile à assurer et la maintenance plus compliquée (chaque dispositif en panne peut mettre en péril les applications).
Remarque 1 : un réseau peer-to-peer ne doit pas être confondu avec une application peer-to-peer qui peut se dérouler dans des réseaux classiques ; des phases d'initialisation centralisées permettent ensuite des échanges point-à-point entre les nœuds.
Remarque 2 : on parle de réseau peer-to-peer dans un cadre fixe et filaire. Pour une technologie non filaire, on parle de réseau ad hoc.

e) Réseaux mobiles

Dans les réseaux, la notion de mobilité recouvre plusieurs domaines :

- les réseaux GSM sont dits mobiles parce que les utilisateurs se déplacent avec leur téléphone et que le service leur est offert (presque) n'importe où. En revanche, il existe une infrastructure fixe (les antennes). C'est aussi le cas des réseaux organisés autour de points d'accès offrant des services (comme un accès à internet) dans des lieux publics, au sein d'une entreprise, etc. Nous parlerons de mobilité des services car c'est la couverture du service qui entraîne la mobilité (i.e. le service est accessible depuis de nombreux endroits).

- les réseaux nomades définis plus haut permettent d'accéder à un intranet depuis différents endroits extérieurs. On parle alors de mobilité des accès au réseau.

- la mobilité des utilisateurs seule ne suffit pas à parler de réseau mobile : par exemple un réseau d'entreprise utilisant des routeurs sans fil n'est pas un réseau mobile car les problématiques sont quasiment les mêmes que celle des réseaux filaires, mise à part la plus grande importance qu'il faut accorder au chiffrement des échanges.

- le terme de réseau mobile est particulièrement adapté dans le cas où tous les éléments composant le réseau sont mobiles. Les dispositifs doivent être suffisamment légers pour être transportés facilement par un individu et employer une technologie de communication sans fil. Ce type de réseau se décline de deux façons en fonction de la gestion des échanges utilisée : soit les messages passent par un dispositif central (mode maître/esclaves) chargé de les transmettre d'un émetteur vers un destinataire, soit les communications se font directement entre les utilisateurs sans intermédiaire, d'égal à égal (ceci sera présenté en détail dans le chapitre 2). Tous les éléments étant mobiles, cela induit une dynamicité de la topologie du résau plus ou moins forte. De nombreuses difficultés découlent de cette mobilité (cf. chapitre suivant).

1.1.2 Description générale des réseaux

Dans cette section nous définissons un schéma d'organisation interne commun à tous les types de réseau. En fonction de cette organisation, nous proposons ensuite un modèle de cycle de vie global des réseaux, puis une classification des points sensibles qu'il faut sécuriser dans tous les cas.

a) Organisation interne des réseaux : plan concret/plan abstrait

Les réseaux sont composés de plusieurs éléments que nous organisons en deux plans (fig. 2) :

- Plan concret. Ce plan regroupe les éléments permettant au réseau de fonctionner, tels que l'infrastructure matérielle (routeurs, serveurs) et logicielle (tables de routage, fichiers centraux), et les dispositifs utilisateurs. C'est le réseau en tant que support de communication.

- Plan abstrait. Ce plan représente l'utilisation du plan concret par un ou plusieurs groupes d'utilisateurs rassemblés autour d'un même but. Il est composé des utilisateurs, des canaux virtuels d'échange[1] et des applications.

Dans la littérature informatique, le terme « réseau » est employé indifféremment pour désigner ces deux plans. Or, les problématiques et les moyens nécessaires pour traiter ces problématiques ne sont pas les mêmes.

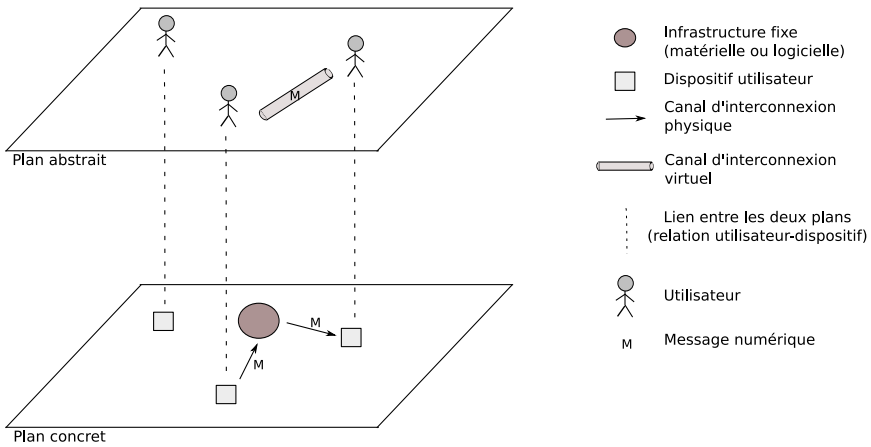


Fig. 2 – Organisation interne des réseaux en plan concret/plan abstrait

b) Cycle de vie d'un réseau

Chacun des deux plans définis précédemment a son propre cycle de vie. Celui du plan concret se résume en deux étapes :

[1] un canal virtuel est une voie d'échange entre des dispositifs, telle que les messages qui y transitent ne sont accessibles que par les extrémités de ce canal. Les éléments relais comme les nœuds intermédiaires du chemin physique ne font que transporter l'information sans pouvoir y accéder. La sécurité des messages est assurée par des moyens cryptographiques.

– la détermination de la politique de fonctionnement du réseau i.e. la manière dont les groupes pourront utiliser le réseau concret. Par exemple, on doit choisir si un ou plusieurs groupes seront acceptés (on peut fixer un nombre limite) afin de déployer le matériel nécessaire ;
– le déploiement du plan concret : cette phase consiste à mettre en place des éléments nécessaires (matériels et logiciels) pour assurer la politique de fonctionnement précédemment établie.

Le cycle de vie d'un réseau sur le plan abstrait comporte deux étapes principales (fig. 3, col. (a)) :
– la détermination de buts : l'utilisation du réseau (concret) a pour origine une volonté humaine. Cette intervention humaine est le déclencheur de toute activité sur le réseau.
– la réalisation de ces buts : les tâches à effectuer pour atteindre ces buts au sein d'un groupe. Il faut pour cela passer par plusieurs étapes :
  – la préparation du groupe : initialisation de paramètres, configurations, etc. ;
  – la formation du groupe : mise en place d'une gestion des accès (pour l'entrée de membre dans le groupe et éventuellement leur sortie) ;
  – la participation au groupe : il faut établir une politique de gestion des nœuds et de leurs communications.

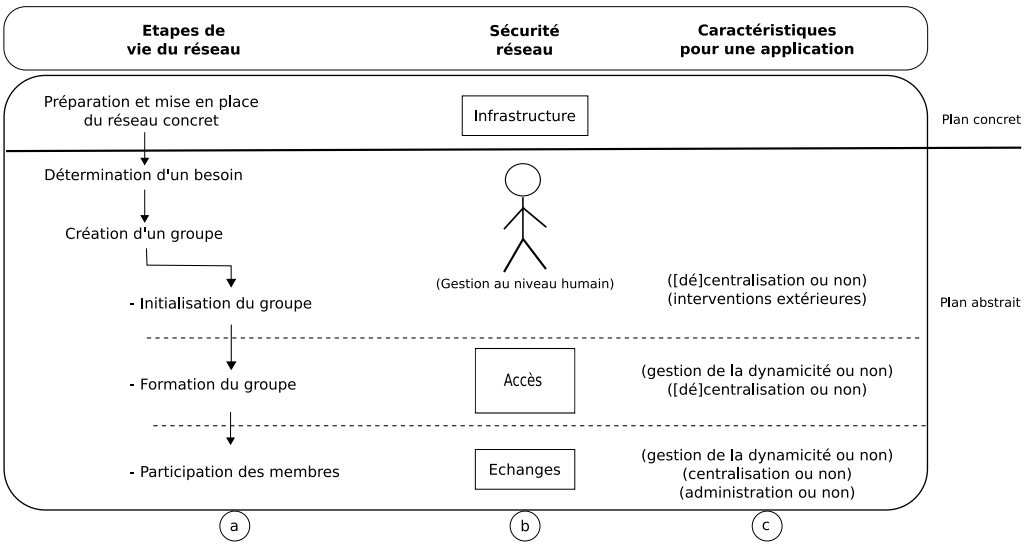


Fig. 3 – Sécurité des étapes du cycle de vie proposé d'un réseau

c) Les points à sécuriser

Nous avons classifié les problèmes de sécurité (fig. 3, col. (b)) selon les deux plans définis précédemment.

Le plan concret engendre les deux prérequis de sécurité suivants :
– il faut que les dispositifs soient personnels, c'est-à-dire utilisables uniquement par leur propriétaire, et en état de fonctionnement ; c'est la *sécurité des dispositifs*.

– il faut que les ressources mises à disposition soient toujours accessibles (donc en état de fonctionnement) ; c'est la *sécurité de l'infrastructure* (matérielle et logicielle).

Au niveau du plan abstrait on peut identifier deux autres prérequis :
– il faut que seules les entités autorisées à profiter d'un groupe le puissent : c'est la *sécurité des accès.*
– il faut que les échanges entre les membres d'un groupe soient sûrs, i.e. qu'ils aient lieu entre les participants autorisés uniquement et qu'ils ne puissent être perturbés d'aucune façon : c'est la *sécurité des échanges.*

1.1.3 Éléments formant le contexte d'évolution d'un réseau

Dans cette section nous présentons les principales caractéristiques des contextes d'évolution des réseaux qui vont influer sur l'élaboration des solutions de sécurité associées. Ensuite, nous proposons une définition du cycle de vie des applications dont chaque étape est sensible à certaines des caractéristiques du contexte déterminé précédemment. Ceci nous servira de base pour élaborer une solution adaptée aux réseaux MANets et qui sera présentée dans la suite de ce document.

Il existe de nombreux liens de dépendance entre les principales caractéristiques des réseaux (fig. 4). Plus on réduit les contraintes des utilisateurs, moins on facilite la mise en place de solutions de sécurité, et inversement. Par exemple, si les utilisateurs sont entièrement libres de leurs déplacements, il est difficile d'utiliser un serveur de contrôle car il ne sera pas possible d'assurer que celui-ci sera accessible par tous à n'importe quel moment (on ne sais pas où sont les nœuds puisqu'ils sont très mobiles).
Les caractéristiques et leurs liens sont représentés figure 4.

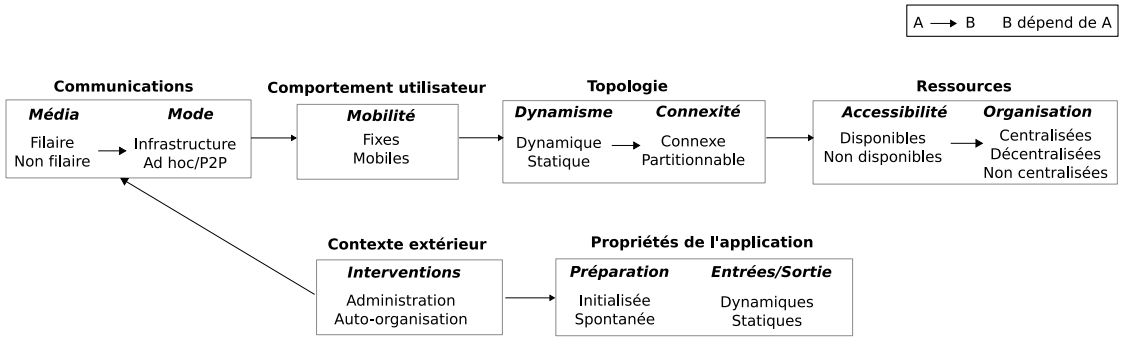


FIG. 4 – Caractéristiques des réseaux

a) Caractéristiques liées aux communications

Les choix du matériel et du mode de communication sont en relation directe avec le comportement des utilisateurs dans le contexte concerné.

Média de communication (fig. 5). Les liens d'interconnexion filaires (fig. 5(a)) sont relativement sûrs par nature. Au sein d'un intranet leur accès est sous la surveillance des administrateurs, le plus souvent dans un bâtiment où l'accès est lui-même restreint aux personnes autorisés. De plus, il est physiquement compliqué d'intercepter les flux passant sur un câble (plus difficile que pour une communication radio). Ces matériels sont donc assez sûrs mais ont pour inconvénient d'être encombrants et lourds à installer, et de contraindre les machines à rester fixes.

Les technologies sans-fil (fig. 5(b)) libèrent les utilisateurs mais, en contre partie, elles présentent plus de risques (nature des ondes, interception possible par tous, etc.) et peuvent ne pas convenir si les besoins de sécurité sont stricts. De plus, les petits dispositifs à technologie sans-fil accentuent les problèmes de disponibilité car ils peuvent disparaître non seulement suite à une décision de l'utilisateur (arrêt de l'appareil) mais aussi à cause de différents facteurs indépendants de toute volonté : nature des ondes, panne de batterie, passage en mode d'économie d'énergie, etc.

Plusieurs associations sont possibles entre les médias et les modes de communications : des réseaux filaires avec des communications centralisées ou peer-to-peer ; des réseaux non filaires en mode infrastructure ou ad hoc.

(a) Filaire (b) Ondes radio

Fig. 5 – Média de communication

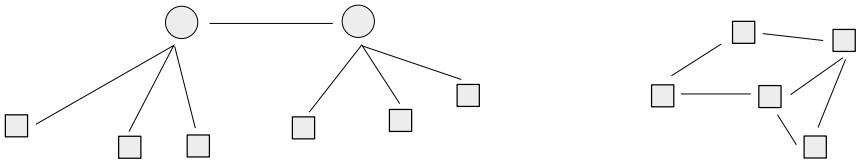
Mode de communications (fig. 6). Dans les modes de communication les plus classiques, les échanges passent par des infrastructures matérielles constituées de *switchs* ou de routeurs (fig. 6(a)). Ces éléments étant dédiés à la réalisation de ces échanges et non accessibles aux utilisateurs, ils permettent un acheminement relativement sûr des paquets. De la même manière, pour les réseaux utilisant une technologie sans fil, un point d'accès peut jouer le rôle d'un intermédiaire par lequel vont passer toutes les communications.

Lien avec les caractéristiques d'administration : ce type d'infrastructure apporte une sécurité au transfert des paquets mais nécessite une installation et une configuration puisqu'il faut se munir de matériels contraignants.
Lien avec les caractéristiques de mobilité : ce type d'infrastructure n'est pas forcément prévue pour être déplacée.

A l'inverse, le mode ad hoc (fig. 6(b)) ou peer-to-peer repose sur une technologie logicielle permettant aux communications de se faire uniquement entre les utilisateurs. Aucune infrastructure physique de routage n'est requise. Ceci permet la mise en place spontanée (i.e. n'importe où, n'importe quand) de la partie matérielle d'un réseau. Les utilisateurs doivent alors procéder eux-mêmes à toutes les opérations d'organisation et de gestion telles

que le transfert des paquets vers leurs destinataires.

Lien avec les caractéristiques d'administration : aucune intervention humaine n'est requise pour l'installation matérielle du réseau.
Lien avec les caractéristiques de mobilité : tous les composants du réseau, y compris les utilisateurs, ont possibilité de se déplacer.



(a) Avec infrastructure (b) Ad hoc

Fig. 6 – Modes de communication

b) Caractéristiques liées aux comportements utilisateurs

Mobilité. Nous avons vu qu'en fonction des technologies de communication choisies les utilisateurs sont plus ou moins libres de leurs déplacements. Dans un contexte sans fil, ils sont en mesure de se servir de dispositifs facilement transportables sans avoir à les relier par un câble aux autres composants du réseau. Un fonctionnement en mode ad hoc augmente cette liberté puisqu'il n'y a plus besoin d'être à proximité d'un point d'accès particulier, les communications intervenant directement entre les nœuds du réseau.
La problématique est qu'on est alors dans l'impossibilité de prévoir quels nœuds seront accessibles par un nœud donné à un moment donné.

Lien avec les caractéristiques topologiques : plus les utilisateurs sont mobiles plus ils risquent de créer de nouveaux liens et d'en rompre d'autres en fonction des portées d'émission des matériels impliqués. Si les postes sont filaires et donc fixes, la topologie reste parfaitement statique.

c) Caractéristiques liées à la topologie

« Dynamicité ». Une topologie est dynamique lorsque des arêtes apparaissent et disparaissent entre les nœuds du réseau. Nous avons vu que la dynamicité était liée à la mobilité des utilisateurs.

Lien avec les caractéristiques de connexité : si la topologie est dynamique, des liens de communication peuvent se rompre et ainsi former des sous-réseau non connexes.

Connexité. Un réseau est connexe si chaque nœud a un chemin possible vers tous les autres. Si une topologie dynamique entraîne l'apparition de sous-groupes (eux-mêmes connexes), on parle de partitionnement du réseau en îlots.

Lien avec les caractéristiques d'accessibilité : lorsque la topologie reste toujours connexe on est sûr que tous les nœuds sont toujours accessibles par tous. Au contraire, si le réseau

est mobile et que la topologie est dynamique, un nœud peut être plus difficile à localiser et à atteindre, et cela peut même devenir impossible s'il se trouve dans un autre îlot.

d) Caractéristiques liées aux ressources

Accessibilité. Un nœud disponible est un nœud que l'on peut atteindre, i.e. avec lequel on peut communiquer pour échanger des informations ou profiter de ses ressources. Exiger la disponibilité d'un nœud permet de l'utiliser comme point central, mais cela restreint la mobilité de son propriétaire.

Lien avec les caractéristiques de centralisation : la disponibilité des nœuds a pour avantage de permettre d'établir une centralisation des ressources et des tâches sur un ou plusieurs points.

Connaissance globale et centralisation. L'intérêt de la centralisation est en général d'investir le ou les points centraux d'une mission particulière (d'en faire des serveurs par exemple). Ceci peut permettre la gestion d'une connaissance globale : on dit qu'une entité a une connaissance globale d'un élément (par exemple les identités des autres nœuds du réseau) lorsqu'elle connaît la valeur de cet élément pour chaque nœud du réseau. Dans le cas contraire, les nœuds ont la seule vision de leurs voisins. Une connaissance globale permet le plus souvent de réaliser des opérations de comparaison entre les éléments récoltés sur chaque nœud. Ceci est utilisé par exemple pour assurer l'unicité des identités, pour choisir le nœud le plus apte à une action, etc.
Un point central peut aussi n'enregistrer aucune information et simplement fournir un service. Il est alors souvent utilisé comme intermédiaire de contrôle par les applications de sécurité par exemple. Le principal inconvénient d'un point central est qu'il doit demeurer accessible par tous les autres nœuds.

e) Caractéristiques d'administration

L'administration d'une application est la réalisation d'une intervention extérieure, pour effectuer des opérations d'organisation ou de contrôle. Cette intervention peut avoir lieu au niveau du plan concret (préparation de l'infrastructure matérielle par exemple) ou du plan abstrait (initialisation des applications, authentification des membres du groupe puis remise d'une clé de participation, etc.).
L'activité d'administration peut être plus ou moins importante. Par exemple, elle peut n'intervenir qu'avant le début du réseau pour la définition et/ou la distribution de certains paramètres. Ces derniers sont alors prédéterminés et statiques (ils n'évoluent pas). En revanche, si l'intervention a lieu pendant l'exécution de l'application, l'ensemble des caractéristiques peut évoluer dynamiquement. En contrepartie, ceci contraint les dispositifs concernés à entrer en communication avec le service d'administration avant de pouvoir réaliser certaines opérations.

Conclusion

Les différents choix de caractéristiques doivent être cohérents par rapport au contexte et au problème à résoudre. Par exemple, pour permettre aux nœuds d'être mobiles, il faut

choisir des dispositifs légers et des supports de communication non-filaires. Dans ce cas, les applications doivent gérer une topologie particulièrement dynamique (apparitions/disparitions de nœuds et éventuels partitionnements) et ne pas compter sur des points centraux.

1.1.4 Cycle de vie des applications

Une application est exécutée dans le cadre d'un groupe de terminaux. Son cycle de vie est composé de plusieurs étapes, chacune dépendant de certaines des caractéristiques de l'architecture mise en œuvre (fig. 3, col. (c)) :

- Initialisation de l'application et préparation des nœuds :
  - centralisée ou non. L'application peut être initialisée sur un nœud central ou de manière distribuée sur chaque nœud individuellement ;
  - administrée ou non. L'initialisation peut reposer ou pas sur la préparation d'un certain nombre de nœuds.

- Entrées/sorties des participants à l'application :
  - gérées dynamiquement ou pas. L'application peut gérer les entrées/sorties des nœuds de manière dynamique (pendant l'exécution même de l'application), ou de manière statique (auquel cas tous les nœuds du réseau doivent être connus avant l'initialisation) ;
  - centralisées ou non : L'application peut centraliser ou pas des informations sur les membres du réseau pour ce qui concerne la gestion des accès (éléments d'authentification, listes d'admission, etc.).

- Participation des nœuds à l'application :
  - gérée dynamiquement ou pas. L'application peut gérer une plus ou moins forte dynamicité de la topologie (des réseaux statiques jusqu'à des partitionnements éventuels).
  - centralisée/administrée ou pas. L'application peut nécessiter ou pas une centralisation ou une administration, par exemple pour sécuriser les communications.

Remarque : dans la littérature, une application est en général dite distribuée si son exécution l'est. Néanmoins, dans la plupart des cas, l'initialisation et la gestion des entrées/sorties des participants sont centralisées ou gérées au niveau humain.

1.2 Les réseaux MANets

Dans cette section nous introduisons les réseaux MANets au travers de leur histoire et des technologies qu'ils emploient, et nous présentons les plus importants domaines de recherche les concernant. Nous étudions ensuite les problématiques spécifiques de ce type de réseaux et qui influent sur le développement des applications qui ont à traiter des aspects liés à la sécurité. Nous considérons enfin plus particulièrement le cas des MANets dits purs.

1.2.1 Historique

Les MANets ont une origine militaire commençant dans les années 80. Leur but premier était d'offrir à l'armée un réseau de communication indépendant d'infrastructures qui nécessitent d'être préparées, empêchent toute action spontanée et peuvent être détruites. Cette technologie permettait en outre aux soldats de préserver leur liberté de mouvement et leur rapidité de déplacement. L'intérêt de tels réseaux pour la vie civile est apparu par la suite. Quelques dates permettront de retracer les étapes principales de leur évolution :

- en 1970 le projet ALOHA [127] crée le premier réseau fonctionnant en mode ad hoc par un médium radio. Le seul traitement d'ordre protocolaire était alors celui des ré-émissions en cas de collision.

- en 1972 naît le projet *DARPA Packet Radio Network* [90] (PRNet). C'est une combinaison d'ALOHA et de CSMA pour partager dynamiquement le canal radio. Un routage basé sur des vecteurs de distance permet d'élargir la portée des échanges qui deviennent alors multi-sauts.

- en 1983 le projet SURAN (*Survivable Radio Network*), toujours développé par la DARPA, étend PRNet. Il propose des protocoles radio plus sophistiqués et des algorithmes réseau gérant plusieurs dizaines de milliers de nœuds. Les dispositifs sont plus petits, peu coûteux et emploient une technologie radio peu puissante.

- en 1987 le projet LPR (*Low Cost Paquet Radio* [63]) propose d'autres améliorations comme des techniques d'étalement de spectre, des protocoles avancés de gestion de réseau et l'utilisation de topologies hiérarchiques basées sur un groupement dynamique afin de s'adapter à la taille des réseaux.

- en 1994 le système *DARPA Global Mobile Information System* (GloMo) [98] adapte des protocoles développés pour Ethernet aux réseaux mobiles sans-fil afin qu'ils puissent se former n'importe où, n'importe quand et avec une topologie très dynamique.

- deux déploiements grandeur nature de réseaux MANets ont été réalisés par l'armée américaine en 1997 avec le projet *Tactical Internet* (TI [90]) ou encore en 1999 avec la *Littoral Battle-space Advanced Concept Technology Demonstration* (ELB ACTD [3]).

- les applications civiles des réseaux MANets ne sont apparues qu'au milieu des années 90 avec l'arrivée des standards 802.11, Bluetooth et Hiperlan présentés de manière plus détaillée dans la section suivante.

1.2.2 Les différentes technologies

Les MANets (*Mobile Ad hoc Networks*) sont des réseaux dont les dispositifs sont, en général, des éléments de petite taille transportables par un individu. Ils utilisent une technologie sans-fil et fonctionnent en mode ad hoc. Ces dispositifs peuvent donc être extrêmement mobiles et suivre tous les déplacements de leurs utilisateurs. La topologie peut alors devenir fortement dynamique. Toutes ces caractéristiques et les problématiques qu'elles

engendrent sont détaillées dans la section 1.1.3 (page 6) de ce document.

Nous présentons dans un premier temps les différentes technologies matérielles employées, puis nous étudions les technologies de communication sans fil utilisées avec ces matériels.

a) Les dispositifs

L'extension des réseaux mobiles sans-fil a été rendue possible par l'apparition sur le marché de dispositifs facilement transportables, car petits et légers, permettant ainsi une réelle mobilité des utilisateurs avec leurs équipements. Ce type de dispositif est apparu dans les années 80 et n'a cessé d'évoluer depuis.

Ordinateurs portables. Ce sont les plus anciens des matériels portables non professionnels. Ils sont relativement encombrants et lourds (jusqu'à 5kg) et leur autonomie est plutôt faible (quelques heures). En revanche, ils sont aujourd'hui très puissants et leurs capacités peuvent égaler celles de PC, i.e. les ordinateurs dits de bureau.

Tablet PC. Ce sont des machines dont la caractéristique principale est leur écran tactil. Elles supportent les mêmes systèmes d'exploitation et les mêmes applications que les PC standards. Par rapport aux portables, elles ont l'avantage d'être plus légères et d'avoir une autonomie plus grande. Elles sont un compromis entre la légèreté du PDA (voir paragraphe suivant) et les capacités d'un portable. Leur principal inconvénient est leur prix plus élevé que celui des portables.
Remarque : de plus en plus, les portables intègrent la technologie des Tablet PC (portable à écran tactil) et ces deux matériels fusionnent.

PDA (Personal Digital Assistant). Les PDA sont des dispositifs extrêmement légers (dont le poids est inférieur à 200g) et qui offre un système d'exploitation proche de celui d'un PC (Windows CE, .NET ou Palm OS) bien qu'ils soient beaucoup moins performants. En revanche, leur autonomie est beaucoup plus grande (jusqu'à plusieurs jours). Malgré leur petite taille, leur utilisation est rendue plus pratique par un écran tactil permettant d'écrire avec un stylet. Cela rend tout de même délicat, voire impossible, la saisie de textes trop longs tels qu'un programme informatique.

Téléphones cellulaires. Les téléphones portables récents offrent les mêmes fonctionnalités que les PDA (ce sont des téléphone-PDA). Ils peuvent donc former des réseaux mobiles les uns avec les autres mais aussi accéder au réseau internet (quasiment) en continu par le biais du réseau GSM ou GPRS. Ces téléphones sont souvent munis d'une technologie comme WiFi (cf. sous section (b)) qui leur permet d'être associés aux autres équipements portables et de partager l'accès à Internet qu'ils fournissent.

b) Les technologies de communication

IEEE 802.11. Le nom générique de cette technologie est WiFi [68] (*Wireless Fidelity*). Normalisée en 1999 par l'IEEE (Institut of Electrical and Electronics Engineers), elle est

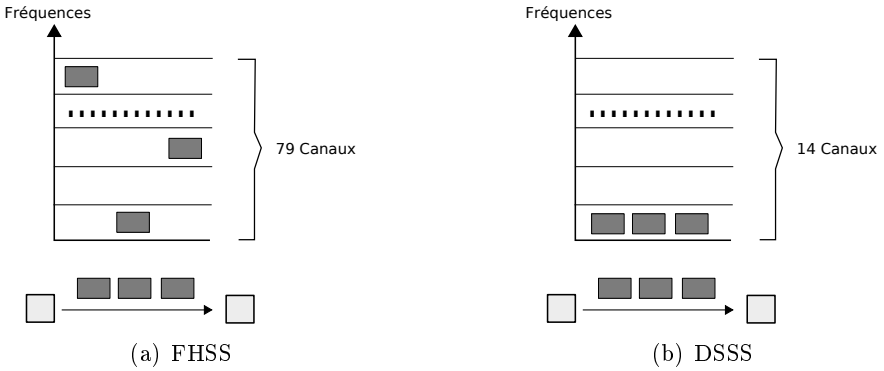
basée sur la norme 802.11.

Il existe plusieurs versions de la technologie WiFi (nommées 802.11x où x est une lettre) qui diffèrent principalement par leurs portées, leurs débits et leurs méthodes de modulation de fréquence (fig. 7) utilisées :

- modulation par étalement de spectre direct (DSSS, *Direct Sequence Spread Spectrum*) : la bande de fréquence est divisée en 14 canaux. Les données ne sont transmises que sur un de ces canaux ce qui rend cette méthode plus sensible aux interférences. Elle utilise la bande de fréquence ISM (*Industrial, Scientific and Medical*) autour de 2,4GHz (2,400-2,4835 GHz). C'est une bande librement utilisable et exploitée par de nombreux protocoles (WiFi, HomeRF, etc). Elle est en outre très sensible à de nombreuses interférences de la vie quotidienne (four micro-ondes, moteurs de certains scooters, etc.).

- modulation par étalement de spectre par saut (*Frequency Hopping Spread Spectrum*), où la bande de fréquence est découpée en 79 canaux. Le canal utilisé pour le transport des messages change très fréquemment. Lorsqu'un paquet n'arrive pas à destination à cause d'une interférence, il est ré-émis avec une forte probabilité sur un autre canal. Cette méthode a pour but minimiser l'effet des interférences. En revanche le débit est moins élevé que pour la modultion DSSS. Cette méthodes utilise aussi la bande de fréquence ISM. DSSS n'apporte rien en terme de sécurité, l'enchaînement des canaux étant normalisés.

- modulation par multiporteuses (*Orthogonal Frequency Division Multiplexing*, OFDM). Dans ce cadre, la bande de fréquence est tout d'abord divisée en 8 canaux de 20MHz eux-même redivisés en 52 sous-canaux. Lors d'une transmission, tous les sous-canaux sont utilisés en parallèle (4 sont réservés à la correction d'erreur). L'algorithme FFT (*Fast Fourier Transform)* est utilisé pour transmettre les messages à travers ces canaux puis pour les recomposer. Cette méthode s'utilise sur la bande de fréquence UNII (*Unlicensed National Informa- tion Infrastructure)* allant de 5,15 à 5,35 GHz.



(a) FHSS (b) DSSS

Fig. 7 – Type de modulation des ondes

WiFi permet de créer des réseaux locaux de type Ethernet mais utilisant une technologie

de transmission des données sans fil. Il est possible de relier des dispositifs sur un rayon de plusieurs dizaines de mètres dans des bâtiments et plusieurs centaines de mètres en extérieur. Ces réseaux sont appelés WLAN (*Wireless Local Area Network*). Les deux modes de communication présentés en 1.1.3(a) (page 6) sont possibles : en mode infrastructure, toutes les communications passent par un dispositif maître (appelé point d'accès) ; en mode ad hoc, les communications se font directement entre les dispositifs, sans infrastructure. L'utilisation principale de cette technologie est le mode infrastructure avec un point d'accès offrant une ouverture vers un ensemble de services (notamment une connexion à Internet).

Bluetooth. Le protocole Bluetooth [30] est défini par le SIG Bluetooth (*Special Interest Group*). Ce groupe a été créé en 1998 par de grandes entreprises du secteur des télécommunications (*Ericsson*, *Nokia*, *Toshiba*, *Intel*, *IBM*) et compte aujourd'hui plus de 2000 membres dont *Agere*, *Microsoft*, *Motorola* ou encore *3com*.

La partie radio Bluetooth utilise la bande ISM avec une modulation par saut de fréquence (FHSS, cf. WiFi).

Bluetooth utilise une organisation de type maître-esclave (fig. 8) entre les dispositifs dans laquelle les esclaves ne peuvent pas communiquer entre eux. Cette organisation est dynamique et automatique, transparente à l'utilisateur. Chaque nœud peut devenir maître ou esclave. Un maître a au plus sept esclaves (on parle de *piconet*), mais le maître d'un *piconet* peut être esclave d'un autre piconet, formant ce que l'on appelle un *scatternet*. De cette manière on peut avoir un fonctionnement pseudo ad hoc : les nœuds communiquent par paires, soit en tant que maître, soit en tant qu'esclave. Une des principales tâches du maître est de déterminer la séquence des sauts (voir FHSS, paragraphe précédent) pour tous ses esclaves.

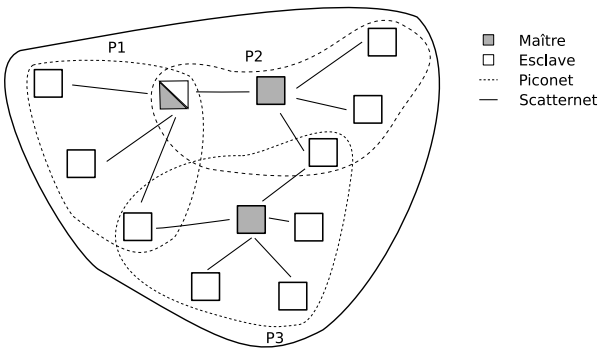


FIG. 8 – Organisation des communications en Bluetooth

Cette technologie est utilisée pour former des PAN (*Personal Area Network*) qui sont des réseaux s'étendant généralement sur une dizaine de mètres. Leurs utilisations sont donc limitées à des communications de portée très courte et les dispositifs sont le plus souvent tous accessibles physiquement par tous les utilisateurs du PAN concerné (il s'agit en général de dispositifs du même utilisateur). L'avantage de cette technologie provient de la très petite taille des puces et de leur faible coût. Elles peuvent s'ajouter à n'importe quel

matériel électronique qui devient alors capable de communiquer. Une des problématiques principale est la consommation relativement forte d'énergie rendant l'autonomie de dispositifs tels que les PDAs ou les téléphones beaucoup plus faible en présence de ces puces, et ce malgré des modes d'économie d'énergie.

HiperLAN (ou *High Performance Radio LAN*). HiperLAN [150] est une alternative au 802.11 ; il s'agit d'un standard européen initié par l'ETSI (*European Telecommunications Standards Institute*) en 1998. Il en existe deux versions ; HiperLAN1 (20 Mbps) et HiperLAN2 (54 Mbps), qui sont compatibles entre elles. Les intérêts et utilisations sont donc exactement les mêmes que ceux de WiFi. HiperLAN utilise la bande de fréquence des 5GHz avec une technique de modulation OFDM (cf. WiFi).

1.2.3 Etat de l'art des recherches dans le domaine des MANets

Dans cette section, nous présentons un aperçu des recherches les plus actives spécifiques aux MANets (il ne s'agit pas d'une liste exhaustive), classées en fonction de domaines plus précis :

- le routage : le projet *Terminodes* [21], datant des années 80, compte parmis les plus importants sur le sujet. De manière plus générale, la mise en place de communications multi-sauts dans un contexte de MANet a nécessité le développement de nouveaux protocoles (AODV [165], DSR [82, 88], OLSR [47], TBRPF [120]). Un aperçu de ces protocoles est présenté dans [1].

- la localisation des nœuds [34, 70, 125, 128] : cette problématique a été entièrement revue dans le contexte de la mobilité. Le plus souvent, ce sujet est étudié en parallèle du routage (pour localiser le nœud à qui on souhaite envoyer un message).

- la gestion de l'énergie [56, 149] : c'est un domaine d'étude très important car les dispositifs dans les MANets ont une autonomie relativement faible et les applications (comme le routage) consomment de l'énergie en quantité non négligeable.

- la diffusion des messages : la plupart des solutions utilisent l'inondation du réseau pour transmettre certains messages (pour l'établissement du routage par exemple). Pour cela, lorsqu'un nœud reçoit un paquet à diffuser, il le ré-émet, ce qui est parfois inutile (par exemple s'il a déjà été ré-émis par un voisin proche). Des études ont donc cherché à limiter le trafic de manière à ce que les nœuds ne ré-émettent que lorsque cela est nécessaire. Plusieurs techniques sont présentées dans [12] ;

- la qualité de service (QoS, *Quality of Service*) [159] : la qualité de service dans les MANets ne peut pas être basée sur les mêmes critères que pour les réseaux filaires (manque de fiabilité des liens, « dynamicité » de la topologie, incertitude sur les temps de réponse, etc.). Dans le contexte des réseaux mobiles, il peut en revanche être important de disposer de mécanismes de QoS permettant de gérer la perte de paquets, de mettre en place des priorités sur les flux, etc ;

- l'interconnexion avec Internet [5] : le but est d'avoir un accès sans fil à Internet où

que l'on soit, à tout moment ;

- les simulateurs de MANets [15, 42, 119, 166] : ils ont pour particularité de devoir prendre en compte de nouveaux paramètres par rapport aux simulateurs de réseaux filaires, comme l'impact de l'environnement sur la portée des ondes (murs), de devoir proposer un comportement réaliste des nœuds relativement à leurs déplacements dans des lieux fermés ou ouverts (modèles de mobilité [28, 85, 113, 163]), etc.

- la sécurité des communications : c'est un sujet particulièrement complexe dans le cadre des MANets. Nous l'étudions en détail dans le chapitre 3.

L'ensemble des études citées cherche à adapter les MANets au mode de fonctionnement et au type d'utilisation des réseaux classiques. D'autres études, en revanche, ont choisi de développer des applications adaptées à la nature des MANets : par exemple, au lieu d'exiger qu'un message arrive à son destinataire dans les plus brefs délais grâce à un mécanisme de routage lourd et nécessitant le respect de fortes contraintes de mobilité, les systèmes DTN (*Delay Tolerant Network* [160]) utilisent un protocole beaucoup moins contraignant permettant un acheminement des paquets mais sans garantie de temps. Pour cela, les dispositifs ont un mécanisme de cache qui stocke les messages et les remet au destinataire s'ils le croisent. Ce mécanisme est entièrement transparent pour l'utilisateur.

1.2.4 Problématiques spécifiques pour le développement d'applications

a) Difficultés dûes aux technologies

Faiblesses du canal de propagation. Les MANets utilisent des technologies de communications sans-fil (cf. 1.2.2(b) page 12) ayant de nombreuses faiblesses influant sur la qualité et l'efficacité des communications :

- le canal de communication utilise des bandes de fréquences sensibles aux interférences avec des éléments physiques du quotidien. Les bruits ou simplement l'affaiblissement du signal à une certaine distance peuvent alors provoquer de nombreuses erreurs, jusqu'à entraîner l'impossibilité de communiquer avec un nœud ;

- le canal de propagation est asymétrique (si un nœud A est dans la portée d'émission d'un nœud B, ce nœud B, lui, n'est pas assuré d'être dans la portée d'émission du nœud A). De plus, la portée de chaque nœud varie dans le temps [4]. Ainsi, si A envoie un message nécessitant une réponse, il n'est absolument pas sûr de la recevoir ;

- la portée des ondes radio n'est pas possible à déterminer exactement à cause des obstables qui peuvent la diminuer ou les réflecteurs (miroir, verre, eau, etc.) qui peuvent l'étendre. Ainsi, contrairement au câbles, il n'y a pas de frontière concrète définissable ou observable. Il va donc être très difficile de connaître la taille d'un réseau et de tous les éléments le composant ;

- il y a beaucoup de problèmes relatifs à l'utilisation du support de communication. Par exemple, en WiFi, l'alternance des émissions mise en œuvre pour ne pas brouiller les messages provoque les problèmes du « terminal caché » (un dispositif croit qu'il

peut émettre alors qu'un autre dispositif qu'il ne voit pas émet déjà) et du « terminal exposé » (un dispositif n'émet pas vers un autre dispositif croyant que le canal est déjà pris alors que ce n'est pas le cas). Ces deux problématiques classiques sont détaillées en [86].

Un réseau sans-fil ne convient donc pas à toutes les applications en raison des faiblesses évoquées ci-dessus. Les applications ne tolérant quasiment aucune faute ou sensible à la latence des échanges ne devraient pas utiliser ce type de technologie.

Ressources énergétiques et puissance. Les dispositifs (cf. 1.2.2(a) page 12) peuvent être classés en utilisant trois grands critères : les ressources dont ils disposent, leur encombrement et leur autonomie. Ces caractéristiques sont dépendantes les unes des autres. En effet, plus la taille et le poids sont réduits, plus les ressources diminuent mais meilleure est l'autonomie. Privilégier la mobilité revient à accepter des ressources moindres, il est donc difficile d'utiliser des applications prévues et optimisées pour des réseaux filaires où les nœuds disposent de ressources beaucoup plus importantes. Revenir à des dispositifs plus encombrants permet de bénéficier de plus de ressources mais l'autonomie des terminaux est beaucoup plus réduite. Les applications devront alors utiliser le moins d'énergie possible. Le problème de l'autonomie est très important et le développement d'applications doit prendre en compte ce facteur pour s'y adapter. Cependant, cela est rendu difficile par la grande hétérogénéité des dispositifs.

b) Difficultés d'ordre topologique

« Dynamicité » de la topologie. La « dynamicité » de la topologie représente une des principales difficultés pour le développement d'applications car celles-ci doivent alors tolérer qu'un nœud avec lequel on était en communication disparaisse et que d'autres apparaissent. Cette « dynamicité » est principalement due à la mobilité des utilisateurs qui peuvent entrer et sortir de la zone de réception d'un autre nœud.
D'autres causes peuvent être à l'origine de la disparition de nœuds telles qu'une fin de vie de batterie, un mécanisme de mise en veille, etc. De plus, les technologies sans-fil sont sujettes à des coupures des liens de communication assez fréquentes qui peuvent intervenir pour plusieurs raisons comme des interférences, des bruits, un obstacle, etc. (cf. faiblesse du canal de propagation). Le fait aussi qu'il soit impossible d'émettre et de recevoir en même temps peut rendre un dispositif invisible (problème particulièrement vrai avec la technologie *Bluetooth*). Enfin, le propriétaire peut aussi tout simplement décider d'éteindre son dispositif.

Partitionnement. La « dynamicité » de la topologie peut entraîner un partitionnement du réseau physique aboutissant à l'apparition de plusieurs sous-réseaux que l'on appelle îlots. La centralisation d'une tâche ou de données est alors impossible puisque tout nœud est inaccessible pour les membres des autres îlots que le sien. Le seul moyen de forcer la disponibilité d'un nœud est de contraindre les utilisateurs à moins de mobilité, i.e. à rester dans un espace restreint.

Asynchronisme et non déterminisme. Les faiblesses du canal de propagation et les changements topologiques fréquents ne permettent pas de prévoir quand aura lieu la réception du message envoyé par un nœud à un autre ; les échanges sont asynchrones et les algorithmes doivent en tenir compte.
De plus, ces algorithmes sont non-déterministes c'est-à-dire qu'il n'est pas possible de prévoir l'ensemble de leur déroulement. En effet, la topologie fortement dynamique ne permet pas, en général, de compter sur une période de stabilité et donc de garantir une configuration finale. De plus, les nœuds étant mobiles et les échanges asynchrones, l'ordre du déroulement des actions est totalement imprévisible.

1.2.5 Cas particuliers des MANets purs

a) Les différents contextes des MANets

Les MANets peuvent être exploités (partie abstraite) dans différents contextes :

1. certains travaux les considèrent dans un cadre très peu mobile avec la possibilité pour certains nœuds de rester disponibles et de permettre une centralisation. Il est alors envisageable d'adapter simplement des applications de type *peer-to-peer* classiques ;

2. beaucoup d'études utilisent les MANets dans un contexte géographique restreint permettant des échanges directs entre les utilisateurs indépendamment du réseau (une réunion par exemple). De nombreuses problématiques sont ainsi résolues, notamment celle de la distribution de la première clé (cf. 1.4.1 page 29) ;

3. d'autres recherches, en revanche, se sont orientées vers les MANets dits purs où ces contraintes sont abandonnées. Le but est d'étendre la spontanéité du plan concret à une spontanéité du plan abstrait (utilisation du plan concret, cf. 1.1.2(a) page 4) en éliminant toute centralisation et administration. Les dispositifs présents appartenant tous à des utilisateurs, il est raisonnable que les tâches soient équitablement réparties entre eux. Ceci entraîne de nombreuses difficultés dans le développement d'applications et les solutions existantes sont basées sur une phase d'initialisation, tous les échanges nécessaires à la suppression de la centralisation se faisant avant l'arrivée des nœuds dans le réseau.

b) Difficultés spécifiques

Sans centralisation, pas de connaissance globale. Une centralisation nécessite qu'un ou plusieurs nœuds soient accessibles et chargés d'une tâche supplémentaire. Ils sont donc en général contraints dans leurs mouvements et doivent éventuellement être préparés. Ceci n'est pas compatible avec les réseaux spontanés et équitables que nous souhaitons promouvoir. Dans notre contexte les MANets ne supportent donc pas la centralisation. Or, sans centralisation, il n'est pas possible d'avoir une connaissance globale d'éléments du réseau et donc de confier à un ou quelques nœuds une tâche particulière dépendant d'une telle connaissance (cf. 1.1.3(d) page 9).

Sans administration, pas de structure de contrôle. L'administration a pour but de contrôler et de gérer les éléments et les évènements d'un réseau. Elle peut être intégrée au réseau ou intervenir à l'extérieur. Il s'agit d'une structure supplémentaire avec laquelle les nœuds sont contraints d'échanger avant de pouvoir participer à une application. Elle ne permet donc pas une utilisation réellement spontannée. L'absence d'administration entraîne d'importantes complications pour ce qui concerne la sécurité puisqu'il n'existe plus de contrôle des utilisateurs. Toutes les opérations doivent être menées par les utilisateurs eux-mêmes. Les données de l'application de sécurité qui réalisent ces contrôles se trouvent alors sur les nœuds utilisateurs qui ne sont pas forcément honnêtes et qui peuvent chercher à modifier le comportement de l'application.

1.3 Introduction aux communications sécurisées

Dans la première section, nous avons vu que les caractéristiques des réseaux pouvaient former des contextes différents influant sur le développement d'applications (page 6). Dans cette section, nous étudions tout d'abord les contextes dans lesquels peuvent avoir lieu des communications. Nous déterminons ensuite les étapes par lesquelles celles-ci passent ces communications et l'influence des contextes (identifiés précédemment) sur chacune de ces étapes.

1.3.1 Contextes des communications

a) Types de protagonistes

Toute communication engage deux protagonistes, un à chaque extrémité, pouvant être de diverse nature : deux utilisateurs quelconques d'un réseau, un utilisateur et un serveur/une application ou encore deux serveurs ou deux applications. Ces différences sont très importantes pour ce qui concerne la gestion des échanges. Lorsqu'il s'agit de s'adresser à un serveur c'est généralement lui seul qui organise les communications. Au contraire, si les échanges n'engagent que des utilisateurs, tous doivent avoir les mêmes droits et les mêmes devoirs, la même charge de travail.
De plus, pour chacune des configurations que nous venons d'indiquer l'utilisation n'est absolument pas la même : les communications entre individus égaux concourrent à un but commun alors que celles entre des clients et un serveur sont des opérations individuelles autour du service proposé (ex. utilisateur-guichet de banque).

Les MANets sont des réseaux collaboratifs ne comptant que des utilisateurs égaux en charge de toute la gestion des communications.

b) Contextes humains

Une intervention extérieure est très utile pour faciliter la mise en œuvre de solutions de sécurité, mais en fonction du contexte humain, une telle intervention n'est pas forcément possible. Dans notre travail nous considérons deux contextes humains dans lesquels peuvent se dérouler les communications :

1. les utilisateurs font partie d'une même organisation humaine, d'un même groupe

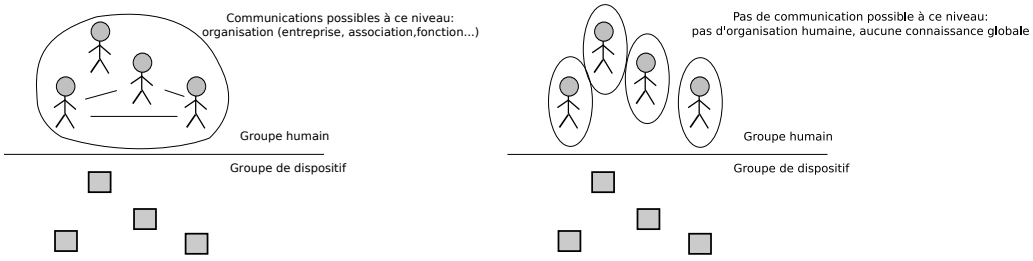
humain (fig. 9(a)). Cette organisation détermine, en général, le but et la frontière du groupe informatique qu'elle met en place.
Exemples :
- membres d'une même entreprise, participants à un colloque, à une conférence, à une réunion, etc ;
- corps de métier qui a besoin de créer un réseau en urgence (police, médecins, pompiers lors d'une catastrophe naturelle, d'un attentat, etc).

2. les utilisateurs ne font pas tous partie d'une organisation commune et n'ont pas forcément d'interaction directe entre eux dans le monde réel (fig. 9(b)).

   Exemples :
   - participants à un forum sur Internet ;
   - rassemblement d'utilisateurs communiquant deux à deux ou en groupe dans une même zone géographique.

Dans le cas où une organisation humaine commune à tous les participants existe, elle peut se charger de toute la gestion des communications. Dans le cas contraire, si une centralisation est possible, ces mêmes tâches peuvent être confiées à un serveur. Si, comme dans notre contexte de MANets purs aucune centralisation n'est possible, il n'existe pas de solution.

Tous les contextes humains sont envisageables pour les MANets. Dans ce travail nous souhaitons apporter une solution prenant en compte chacun de ces contextes. La difficulté provient des cas où il n'y a ni centralisation, ni administration, ni planification possibles.



(a) Avec lien dans le monde humain (b) Sans lien dans le monde humain

FIG. 9 – Contextes humains

c) Contexte lié au plan concret

Les réseaux dédiés (fig. 10(a)). Ce sont les intranets les plus classiques dans lesquels la partie concrète du réseau est utilisée par les membres d'une seule et même organisation. Les termes « réseau » et « groupe » sont alors confondus puisque l'intégralité du réseau est

entièrement dédiée à un même groupe humain. Dans ce cas de figure, la raison d'être du groupe précède la création du réseau, il en est même l'origine. C'est un administrateur qui met en place l'infrastructure matérielle et logicielle nécessaire et qui s'occupe de la gestion de la sécurité.

Les réseaux non dédiés (fig. 10(b)). Dans ce type de réseaux, le plan concret pré-existe à toute intention de groupe. Plusieurs groupes et buts peuvent y coexister. Dans ce cas de figure, le groupe et le réseau sont distincts. Par exemple, le réseau Internet est une infrastructure (matérielle et logicielle) utilisée par des groupes avec des buts différents et gérée par des organisations indépendantes les unes des autres. En revanche, dans le cas précis d'Internet, ces organisations sont elles-mêmes contrôlées par une administration supérieure (à l'origine du plan concret). Cette dernière est composée d'organisations hiérarchisées, les autorités les plus proches des utilisateurs étant les fournisseurs d'accès à Internet.

Les réseaux MANets font partie des réseaux non dédiés.

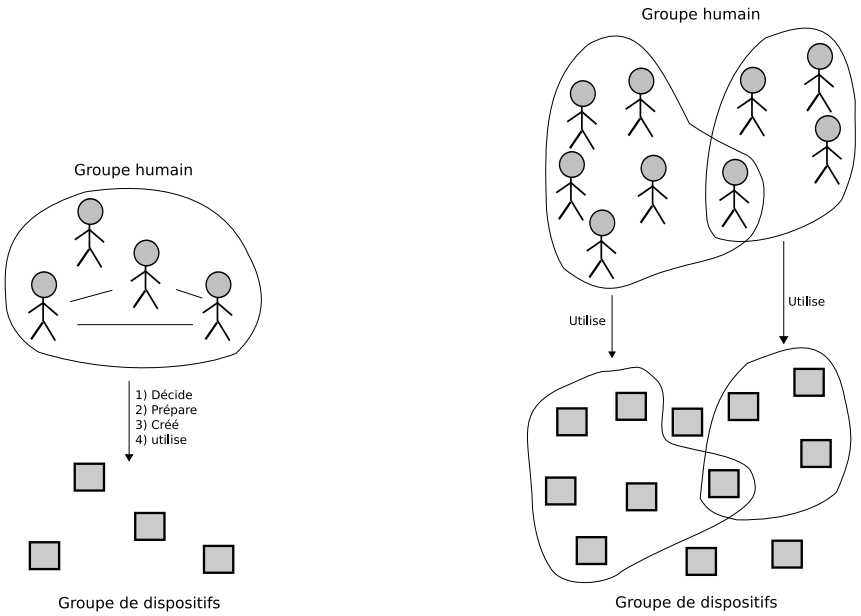
En effet, la technologie de communication radio des MANets fait que tout dispositif dans un certain rayon accède physiquement au réseau. L'établissement du réseau matériel ne demandant aucune préparation, c'est comme s'il pré-existait. Ainsi, tout comme pour Internet, plusieurs organisations humaines peuvent utiliser le même réseau concret et plusieurs buts peuvent coexister. En revanche, aucune autorité ne peut contrôler l'accès au réseau physique (pas de fournisseur d'accès). La différenciation entre les groupes, leur protection les uns vis à vis des autres, etc., sont alors très problématiques puisqu'aucune administration globale n'est présente pour procéder aux contrôles. Les solutions de sécurité doivent obligatoirement passer par des dispositifs utilisateurs qui ne sont pas forcément tous honnêtes.

1.3.2 Etapes nécessaires pour la réalisation des communications

Nous considérons qu'une communication entre deux éléments se compose de trois phases. En effet, il faut :
- déterminer la raison de la communication : quel est le but de l'échange ;
- déterminer la cible : quel est l'interlocuteur ;
- réaliser l'échange à proprement parler.

Nous étudions l'influence des contextes déterminés précédemment sur chacune de ces étapes.

Détermination de la raison de la communication. Tout échange au plan abstrait est amorcé par une volonté humaine dans un but précis. Il faut donc que les individus aient de bonnes raisons de communiquer.



(a) Réseaux dédiés (b) Réseaux non dédiés

FIG. 10 – Contextes des groupes en fonction du plan concret

Nous distingons deux cas de figure selon le contexte humain :

1. les personnes font partie d'une même organisation humaine qui leur donne une raison d'échanger (par exemple, une entreprise et doivent réaliser un travail particulier) ;
2. les utilisateurs n'ont pas de lien au niveau humain et n'ont donc pas a priori de raison d'échanger.

Dans le cas où une centralisation est possible, les utilisateurs se rassemblent autour d'un point central affichant les caractéristiques d'un groupe (identité, but, etc.). C'est le cas d'un serveur sur Internet gérant par exemple des sujets de discussion. Si la centralisation est impossible il n'existe pas de solution permettant aux utilisateurs de « publier » le but des communications qu'ils souhaitent établir.

Dans les MANets purs non planifiés, les utilisateurs ne se connaissent pas forcément entre eux et n'ont pas d'interaction directe au niveau humain. Ils n'ont donc pas, à la base, de raison d'échanger. Il est nécessaire que des individus ou des groupes puissent exprimer leur but afin d'éveiller chez les utilisateurs voisins des raisons de communiquer.

Détermination de l'interlocuteur. Un émetteur doit aussi déterminer avec qui il souhaite communiquer. Pour cela il faut qu'il sache quels sont ses interlocuteurs possibles et qu'il connaisse pour chacun certaines de ses caractéristiques : l'identité d'une personne, sa fonction au sein du groupe (directeur de l'entreprise, président de l'association, etc.), son métier, ses goûts, ses connaissances dans un domaine, etc. De la même manière que pour la détermination de la raison des communications, dans les solutions existantes ce besoin

est géré soit par l'organisation humaine, soit par un ou plusieurs serveurs dédiés qui centralisent les données. Par exemple dans les entreprises, quelqu'un est chargé de répertorier l'ensemble des employés et leur fonction ; la gestion des informations sur les membres du groupe est donc assurée par l'organisation humaine. Sur Internet, des serveurs peuvent centraliser des caractéristiques entrées par les participants.
Si un point central n'existe pas il est possible de prendre connaissance des qualités d'un interlocuteur voisin en échangeant directement avec lui. La question est alors de savoir si les données reçues sont « correctes » ou non.

Dans les MANets purs non planifiés, les individus doivent eux-mêmes présenter leurs caractéristiques puisqu'elles ne peuvent pas être centralisées.

Echange des données. Une fois la raison des communications et l'interlocuteur déterminés, il est possible de procéder aux échanges. Il s'agit alors de traiter les problématiques habituelles : confidentialité, intégrité, authentification et non répudiation. Une fois la première clé échangée (cf. 1.4.1 page 29), il existe un nombre de protocoles et d'applications permettant de mettre en place ces sécurités. Dans les solutions classiques, la problématique de la diffusion de la première clé est résolue par des interactions au niveau humain.

Dans les MANets purs sans planification aucune solution existante n'est exploitable car elles reposent toutes sur l'utilisation d'une organisation humaine commune et/ou sur une planification.

1.4 Sécurité de la partie abstraite : un état de l'art

La partie abstraite d'un réseau est l'exploitation de sa partie concrète au profit de groupes d'utilisateurs unis par une même autorité ou un même but. Sécuriser cette partie consiste donc à permettre et protéger les différentes étapes de la vie des groupes : leur création, leur constitution (entrées/sorties des nœuds) et les échanges entre leurs membres.

Dans cette section, nous présentons tout d'abord les outils utilisés pour sécuriser cette partie abstraite. Nous détaillons ensuite les problématiques et les solutions existant pour chacune des étapes à sécuriser. Pour cela, nous différencions (en deux sections) la sécurité des accès à un groupe (création et constitution) et la sécurité des échanges en son sein.

Remarque 1 : la création des groupes n'est pas réellement une problématique des réseaux classiques car c'est le niveau humain qui s'en occupe. Une discussion à ce propos aura lieu dans le troisième chapitre de ce document.
Remarque 2 : dans cette section nous présentons des solutions existantes, mais il ne s'agit pas encore de faire de propositions.

1.4.1 Outils cryptographiques et informatiques

La cryptographie [135] est la science qui étudie le chiffrement des données qui peut être utilisé dans différents buts :

- il peut servir à protéger des données locales contre toute lecture illégale (directement sur la machine concernée ou depuis l'extérieur par le réseau).
- il est nécessaire à la mise en place de canaux virtuels sécurisés utilisés pour réaliser des échanges lors d'une conversation entre utilisateurs ou pour mettre en place un protocole d'authentification.
- il permet aussi la signature de messages pour en authentifier l'émetteur. Par exemple, une autorité peut signer des informations et ainsi prouver qu'elle les a validé.

La cryptographie intervient donc dans la sécurité des accès (authentification des entités) et dans la sécurité des échanges (chiffrement des conversations et authentification des protagonistes).

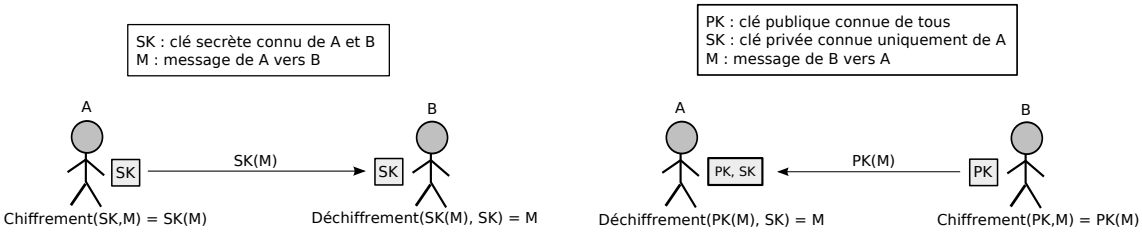
a) Le chiffrement des données

Les principaux algorithmes. Le chiffrement des données est réalisé par des algorithmes particuliers qui peuvent publics ou privés. Les algorithmes publics se basent sur des éléments secrets. Au contraire, les algorithmes secrets n'utilisent que des données publiques. Cette dernière méthode a pour avantage d'éviter le problème de l'échange d'informations secrètes. Cependant, d'après le principe de Kerckhoffs [92], la sécurité doit toujours reposer sur le secret de la clé et non pas sur celui de l'algorithme. En effet, tout secret a des failles et risque d'être découvert. Si c'est le cas, il faut que cela cause le moins de dommages possibles et que le secret en question soit facile à changer. Si la sécurité d'un système repose sur le secret d'un algorithme, la découverte de celui-ci fait tomber la sécurité de tout le système. Par contre, si une clé de chiffrement est découverte, seuls les échanges utilisant cette clé sont compromis. Il suffit alors de la remplacer sans intervenir sur le reste du système.

Il existe deux types d'algorithmes de chiffrement utilisés aujourd'hui [73] :

- les algorithmes à clés symétriques (fig. 11(a)), où les clés de chiffrement et de déchiffrement sont telles que l'une permet de trouver l'autre (le plus souvent c'est une seule et même clé). Ces clés doivent être uniquement connues des entités autorisées à accéder aux données qu'elles protègent. Les algorithmes à clés symétriques ont l'avantage d'être rapides. En revanche, pour des échanges privés deux à deux (cf. 1.4.3 page 42) il faut partager une clé différente pour chaque paire de nœuds. Les utilisateurs doivent donc créer, distribuer et stocker autant de clés qu'ils ont d'interlocuteurs. Ces algorithmes sont principalement utilisés pour assurer la confidentialité. Une clé symétrique peut aussi servir de secret de groupe si tous les membres de ce groupe, et seulement eux, la possèdent.

- les algorithmes à clés asymétriques (fig. 11(b), où les clés de chiffrement et de déchiffrement sont différentes[1]. L'une des clés est publique et l'autre est privée (elle doit absolument rester secrète) tel qu'il est impossible de deviner (au moins) la clé privée à partir de la clé publique. L'avantage est qu'une seule paire de clés est nécessaire pour communiquer avec tous ses interlocuteurs. En revanche, la génération d'une paire de clés asymétriques demande beaucoup de puissance de calcul et les chiffrements et déchiffrements sont lents.



(a) Chiffrement avec clés symétriques (b) Chiffrement avec clés asymétriques

Fig. 11 – Chiffrement des données

Algorithmes de hachage. Les algorithmes de hachage [73] ne sont pas à proprement parler des algorithmes de chiffrements mais ils ont de nombreuses applications cryptographiques. Ce sont des fonctions à sens unique qui transforment un message en un « haché », c'est-à-dire en un résumé du message de taille beaucoup plus petite. En cryptographie, on utilise principalement les fonctions de hachage pour construire des MAC (*Message Authentification Code*) pour réduire la taille d'un message à signer, ou les fonctions de hachage pour calculer des MDC (*Message Detection Code*) qui permettent de vérifier l'intégrité d'un message reçu (la moindre modification dans un message change le résultat du hachage de celui-ci).

b) Les mécanismes de signatures

On parle de signature s'il est possible d'identifier le nœud ayant procédé à ce chiffrement (fig. 12). Par exemple, dans les systèmes asymétriques, les données sont chiffrées avec la clé privée et le récepteur les déchiffre avec la clé publique associée dont il connait le propriétaire (c'est une clé authentifiée).
Remarque : dans les systèmes symétriques, si la clé est partagée entre deux éléments seulement (clé de session, cf. 1.4.3 page 42) la signature est automatique puisqu'on reconnaît le seul autre nœud qui connait la clé secrète.

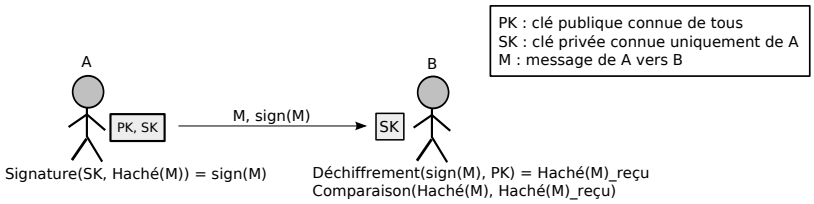
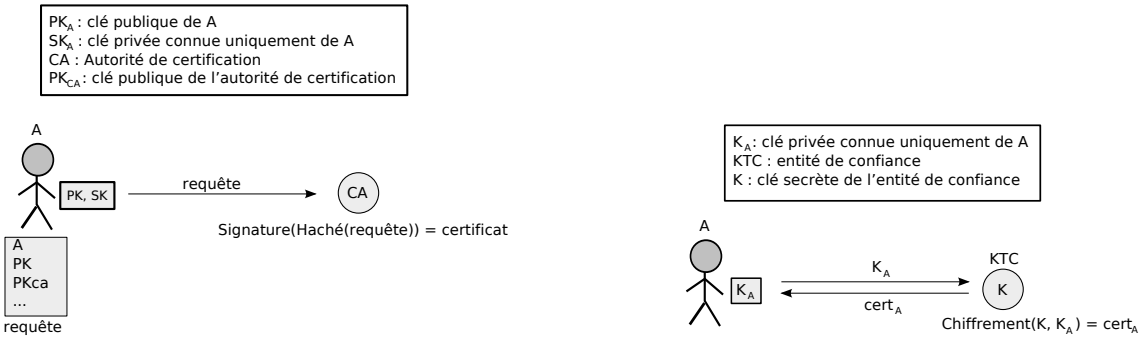


Fig. 12 – Signatures numériques

[1] Le rôle des clés est interchangeable mais on chiffre avec la clé publique pour la protection des données et avec la clé privée pour leur signature (cf. section sur les mécanismes de signature.

c) Les mécanismes de certification

De manière très générale, on peut dire qu'un certificat informatique est un document validé par une entité de confiance (cf. sous section (d) suivante), associant la clé d'un utilisateur à des informations qui lui sont personnelles. Le terme est principalement employé pour les systèmes à clés asymétriques que l'on nomme PKI (*Public Key Infrastructure*) où la clé publique est associée à l'identité de son propriétaire (fig. 13(a)). Ce mécanisme existe aussi pour les systèmes à clés symétriques [89, 108] où c'est la clé secrète d'un nœud qui est associée à l'identité de celui-ci.



(a) Public key certificate (b) Secret key certificate

FIG. 13 – Mécanisme de certification en fonction du type de clé

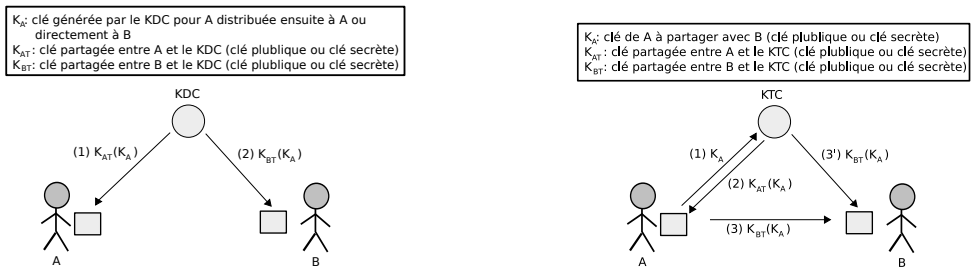
d) Les autorité de confiance

Présentation. La gestion des clés (création, validation, distribution, etc.) peut reposer sur des entités dites de confiance. Ce sont des « autorités » si leur compétence est reconnue par l'administration humaine du groupe. On parle alors d'entités tierces de confiance (TTP : *Trusted Third Party*). Elles peuvent avoir plusieurs rôles (fig. 14) :

- un KDC (*Key Distribution Center*) est chargé de générer et de distribuer des clés de session à la demande des nœuds utilisateurs. La distribution se fait de façon sécurisée en utilisant un canal chiffré différent entre chaque nœud et le KDC. Pour un canal donné, le chiffrement se fait avec une clé publique du nœud utilisateur ou avec une clé secrète que ce nœud partage uniquement avec le KDC.
- un KTC (*Key Translation Center*) ne génère pas la clé mais se charge simplement de la distribution sécurisée des clés de session.
- un CA (*Certificate Authority*) peut être un KDC ou un KTC mais il a en plus pour rôle de signer des certificats (cf. 1.4.1(e) page 28). La signature se fait avec la clé privée de cette autorité. Tous les utilisateurs doivent connaître la clé publique associée pour déchiffrer le certificat (cf. paragraphe précédent).

Ces TTP peuvent se présenter sous trois formes différentes (fig. 15) :

- une autorité *in-line* (fig. 15(a)) appartient au réseau et sert d'intermédiaire direct aux communications entre deux nœuds.
- une autorité *on-line* (fig. 15(b)) appartient au réseau mais n'intervient pas directement dans une conversation entre deux nœuds. Elle possède simplement des données



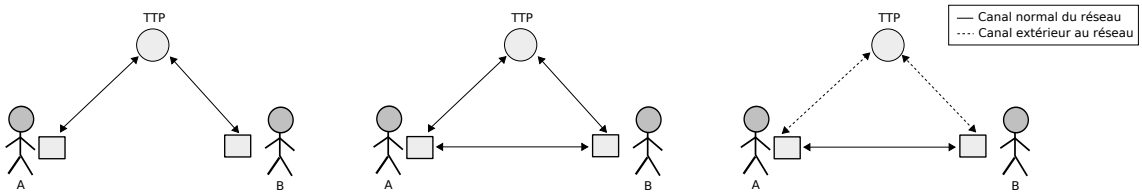
(a) KDC (b) KTC

Fig. 14 – Les différentes autorités de confiance

nécessaires aux communications (comme des certificats ou d'autres éléments publics) qu'au moins un des deux protagonistes de l'échange doit récupérer. Ensuite, les deux nœuds peuvent communiquer entre eux sans intermédiaire.

– une autorité *off-line* (fig. 15(c)) est extérieure au réseau. Avant tout échange entre deux nœuds, l'un d'entre eux au moins doit récupérer, auprès de cette autorité, des données nécessaires par un canal différent de celui du réseau.

Ces autorités de confiances sont décrites précisément en [135].



(a) in-line (b) on-line (c) off-line

Fig. 15 – Implication des TTP

Remarque : ces TTP doivent être initialisées, administrées et accessibles par tous les utilisateurs. Leur mise en place et leur maintien sont donc très contraignants et ne conviennent pas à toutes les configurations de réseaux.

Partage du rôle d'un CA. Pour pallier le problème de disponibilité d'un unique serveur de certification, une méthode a été développée permettant de partager la tâche sur des dispositifs utilisateurs, et ce sans perte de sécurité. Pour cela, les nœuds utilisateurs n'étant pas, de fait, sûrs, il est dangereux de laisser à un seul d'entre eux l'entière responsabilité de la tâche. La solution proposée est donc que plusieurs nœuds collaborent pour accomplir cette tâche. Ainsi, un nœud utilisateur malveillant ou corrompu, seul, ne peut rien faire et ne compromet pas à lui seul la sécurité globale du réseau.

La mise en place d'une telle solution utilise une technique appelée *Threshold Secret Sharing* (TSS) qui consiste a appliquer à la clé privée globale - de certification - le principe du secret partagé de Shamir [137] (fig. 16) : à partir d'une clé unique on créé n clés

partielles. Seules k $<$ n clés partielles sont nécessaires pour recréer la clé complète. Ainsi, chaque serveur possède une clé privée partielle et peut procéder à une signature partielle d'un document. Pour reconstituer une signature complète du document, il faut donc au moins k signatures partielles de celui-ci. En général, une entité de confiance appelée *Trusted Dealer* (TS) établit la clé globale puis calcule et distribue les clés partielles initiales.
Pour s'affranchir de tout administrateur, l'initialisation peut aussi être réalisée collectivement par les membres du groupe selon la méthode du *Joint Secret Sharing* [122] (les membres calculent collectivement les secrets partiels à partir d'une valeur aléatoire commune). L'inconvénient de cette méthode est que cette initialisation est assez lourde du point de vue du nombre de messages à échanger et qu'elle nécessite une concertation entre les utilisateurs.
Du point de vue de la sécurité, des travaux ont été menés sur la validation des clés partielles : il s'agit de vérifier qu'elles ne proviennent pas de nœuds malveillants et qu'elles permettent bien de reconstruire la signature complète. On parle de *Verifiable secret sharing* [45, 64].

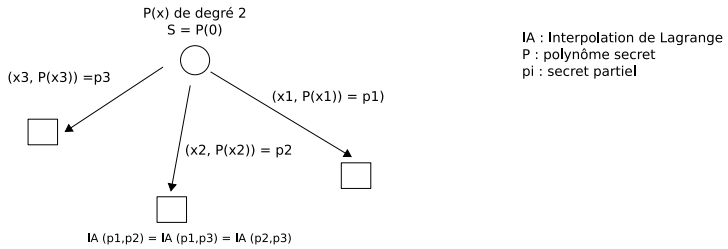


FIG. 16 – Shamir Secret Sharing

> De nombreuses recherches ont cherché à améliorer cette méthode. Par exemple, pour permettre d'adapter le seuil en fonction de la taille du réseau, il existe des méthodes avec des seuils dit dynamiques, dans lesquels k est alors une fraction de la taille courante du groupe [36, 67].

Si la distribution est partielle, plusieurs dispositifs (ceux qui ont une partie de secret) ont une tâche plus importante que le reste des nœuds et leur localisation est difficile. Ce n'est plus le cas si la distribution est totale (sur tous les nœuds), par contre chacun devient alors une cible potentielle et le nombre de nœud qu'il est nécessaire de compromettre pour corrompre le système est plus facile à atteindre. De plus, chaque nouvelle entrée nécessite le calcul et la distribution d'une clé partielle au nouveau membre.

e) L'authentification des clés

Les clés cryptographiques servent à chiffrer et déchiffrer des données au nom d'une entité (un ou plusieurs individus ou processus). L'authentification d'une clé est l'association de la clé à l'identité de son possesseur.

Supports d'authentification. Il existe plusieurs types de support d'authentification : Les certificats (cf. 1.4.1 page 24, fig. 13). Ce sont des documents contenant la clé publique

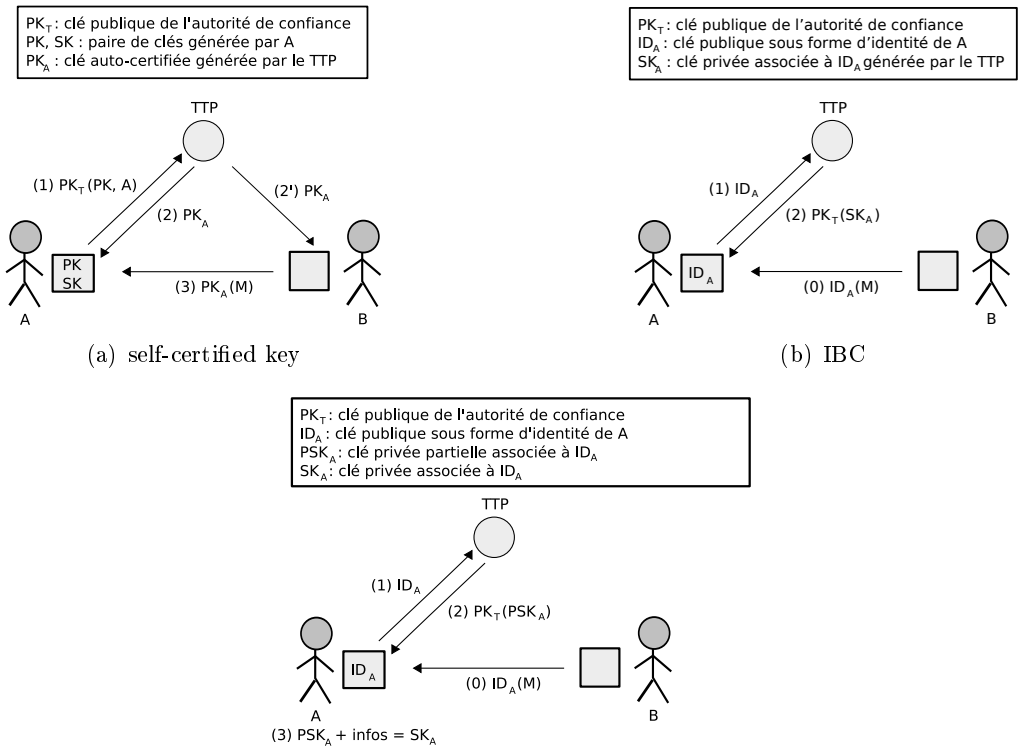
- d'un système de clé asymétrique - et l'identité de son propriétaire. D'autres informations peuvent être ajoutées telles qu'une durée de validité du certificat, un nom d'organisation, etc. Cette association doit être validée par une entité qui est aussi chargée de fournir la preuve de cette validation en signant le certificat (chiffrement avec sa clé privée permettant un déchiffrement avec sa clé publique).

Clé authentifiée ou auto-certifiée (fig. 17(a)). Dans l'approche présentée dans [71] la clé contient elle-même l'identité de son possesseur. Une autorité de confiance génère, grâce à un algorithme secret, la clé auto-certifiée à partir de la clé publique et de l'identité fournie. L'intérêt de cette méthode par rapport aux certificats classiques est d'économiser de l'espace mémoire.

Identity Based Cryptosystem, IBC [27] (fig. 17(b)). Dans ce type de solution, la clé publique est une chaîne de caractères lisible par un humain (une adresse mail, une identité ou un numéro connu, etc.). La clé publique peut alors être échangée au niveau humain et non uniquement via le réseau. Ceci a pour avantage de ne plus avoir besoin d'accéder à une autorité *in/on-line* pour récupérer la clé publique d'un utilisateur. En revanche, une autorité demeure nécessaire pour générer et distribuer la clé privée associée. Il est possible de décentraliser le rôle de cette autorité comme proposé dans [93] (voir 1.4.1(d) page 27). Ce type de solutions présente plusieurs points faibles. Tout d'abord la distribution d'une clé privée par le réseau est très risquée et l'utilisation d'un canal parallèle représente une contrainte importante. De plus, il est toujours préférable qu'une clé privée soit connue de son seul propriétaire. Enfin, cette méthode ne respecte pas le principe de Kerckhoff (cf. 1.4.1 page 24) puisque pour que quiconque ne puisse pas générer la clé privée associée à une clé publique, l'autorité de certification utilise une fonction prenant en paramètre une clé secrète.

Certificateless Cryptography, CL-PKC (fig. 17(c)). Cette approche présentée dans [2] a pour but d'améliorer les systèmes de type IBC afin que les autorités de confiance n'aient plus la clé privée des nœuds du réseau. Pour cela, ces autorités ne produisent plus qu'une clé secrète partielle qu'ils distribuent à leur propriétaire. Celui-ci complète la clé partielle avec des informations privées et forme sa clé privée.

Problématique de la première clé. On voit bien, avec ces outils, que la problématique principale est la distribution de la toute première clé, celle qui permettra de mettre en place le premier canal d'échange sécurisé. S'il y a un administrateur, la clé peut être donnée par celui-ci à un utilisateur à son entrée dans le réseau. C'est un canal parfaitement sécurisé. Pour les solutions dans lesquelles on ne souhaite pas avoir d'administrateur il faut que les nœuds aient accès à un autre canal parallèle d'échanges sûr : échange vocal ou visuel, lien infrarouge, etc. Par exemple, les clés publiques des CA sur Internet (comme *Verisign*, *GTE CyberTrust*, *Thawte* ou *Entrust*) sont très souvent directement fournies avec les navigateurs *web*. Le problème est alors de garantir la nature originale du logiciel (ce qui est difficile pour des logiciels pris dans un environnement peu sûr comme internet).



(a) self-certified key

(b) IBC

(c) CL-PKC

FIG. 17 – Authentification des clés

1.4.2 Sécurité des accès

Sécuriser les accès au réseau c'est s'assurer que, à tout moment, les deux conditions suivantes sont respectées :

- tous les utilisateurs présents dans un réseau ont bien la permission d'y être ;
- toutes les ressources du réseau sont bien utilisées par des individus qui y sont autorisés.

La procédure permettant l'accès d'un utilisateur à un groupe comporte plusieurs étapes (fig. 18) :

- l'obtention de l'autorisation de participer au groupe. Dans toutes les solutions existantes, ceci est décidé au niveau de l'organisation humaine qui sous-tend le groupe. Nous ne détaillerons donc pas ce point.
- la mise en place des paramètres de participation de l'individu (identifiant, authentifiant d'accès, statut au sein du groupe, permissions, etc) avant son entrée.
- la phase d'authentification des candidats.
- l'admission des utilisateurs autorisés.
- la gestion des révocations des utilisateurs qui ne font plus partie du groupe.

Remarque : la phase d'authentification et d'admission auprès d'un serveur n'est pas obligatoire si les informations fournies aux nouveaux membres après autorisation suffisent. Par

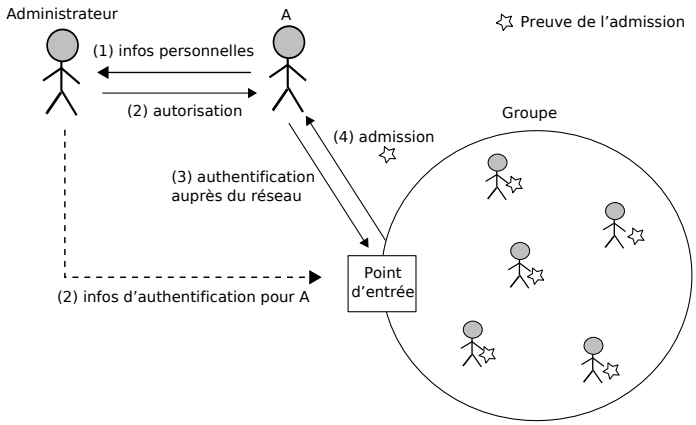


Fig. 18 – Étapes de la gestion des accès à un groupe

exemple, si l'administrateur remet un certificat dédié à un groupe unique, les communications avec un autre membre pourront avoir lieu après une authentification directe auprès de l'interlocuteur. L'inconvénient de cette méthode est que le certificat ne peut être utilisé que pour entrer dans un seul groupe. S'il existe de nombreux groupes auxquels on souhaite participer, la gestion d'un grand nombre de certificats pourrait devenir difficile.

a) Autorisation

Les paramètres utilisateurs sont les données nécessaires à une entité pour participer à un groupe après y être entré. Ces paramètres sont déterminés avant l'accès au groupe, en général par l'entité qui donne l'autorisation d'entrée. Au minimum, il faut gérer l'identité et les droits d'accès de chaque membre.

L'identification. L'identification est l'association d'un identifiant (nom, numéro, etc) à une entité, ici un nœud. L'identification n'est pas une obligation si les communications se font anonymement et sans continuité dans le temps (si on ne souhaite pas échanger de nouveau avec un même nœud). En revanche, elle devient nécessaire pour une conversation privée avec un ou plusieurs nœuds choisis spécifiquement pour une raison précise.

Les différents types d'identifiants. La nature d'un l'identifiant est fonction du niveau d'utilisation suivant :

- Le niveau utilisateur : c'est à l'utilisateur que l'on attribue un identifiant. Classiquement, un pseudonyme est simplement proposé ou choisi par l'utilisateur lui-même. Il est plus pratique d'utiliser un identifiant humainement facile à lire et à mémoriser. Une clé cryptographique, un simple nombre ou même une longue chaîne de caractères peuvent être trop complexes pour un utilisateur s'il a besoin de « reconnaître » les identifiants de ses interlocuteur ou de se rappeler son propre identifiant.

- Le niveau applicatif : ce sont des instances d'applications qui doivent se reconnaître entre elles. L'identifiant n'a donc plus besoin d'être lisible par l'utilisateur puisqu'il

sera traité par une machine. Il peut donc s'agir de n'importe quelle chaîne de caractère ou nombre généré par la machine.

- Le niveau matériel/réseau : il peut aussi être nécessaire de s'identifier auprès d'une couche réseau ; c'est par exemple le cas du système TCP/IP où chaque matériel (carte réseau) possède une adresse IP. Ces adresses sont, soit choisies statiquement, soit attribuées dynamiquement par un serveur *dhcp*.

Problématique de l'unicité. Pour individualiser les nœuds, il faut attribuer à chacun d'entre eux un identifiant unique. Cette différenciation stricte de chaque entité permet que les messages arrivent aux seuls éléments ciblés. On ne peut être sûr qu'un élément est unique que de deux manières :

- par comparaison. Les comparaisons sont possibles si tous les identifiants sont stockés sur un ou plusieurs points centraux ou lorsqu'on est sûr de pouvoir interroger tous les éléments du groupe à tout moment. Chaque nouvel identifiant (demandé par un nouveau membre) est comparé avec ceux déjà utilisés. S'il est attribué, un autre devra être proposé.

- par unicité naturelle. L'absence de centralisation, d'administration et une grande mobilité des nœuds rendent difficiles les opérations de comparaisons d'un nouvel identifiant avec ceux déjà utilisés. La seule solution est alors d'avoir un identifiant unique par nature. Une clé cryptographique, par exemple, est en théorie (avec une très forte probabilité) unique par construction. Pour que l'unicité soit garantie il faut, en outre, protéger l'identité contre le vol. On peut aussi empêcher son possesseur de la distribuer volontairement (par exemple pour s'en servir sur deux dispositifs différents au même moment).

Droits d'accès aux ressources. Il faut ensuite déterminer les droits d'accès de chacun aux ressources et aux données. Plusieurs politiques existent [57] dont voici les principales :

- Contrôle d'accès obligatoire (MAC, *Mandatory Access Control*) : les droits d'accès sont imposés par le système (pas par le propriétaire d'un objet).
- Contrôle d'accès discrétionnaire (DAC, *Discretionnary Access Control*) : les autorisations sont décidées par ceux qui ont les droits nécessaires (en général les propriétaires).
- Contrôle d'accès à base de rôles (RBAC, *Role-Based Access Control* [61]) : les droits d'accès sont définis par l'administrateur en fonction de rôles et non plus d'individus. Les rôles sont attribués à chaque utilisateur par ce même administrateur. L'avantage est qu'il n'y a pas besoin de modifier les droits d'accès à chaque changement de composition du groupe.

b) Authentification d'un utilisateur auprès du groupe

Pour accéder à un groupe, l'utilisateur doit prouver qu'il y a été autorisé. Cette vérification se base sur la possession d'un *authentifiant*, sur la distribution aux serveurs d'authentification des informations nécessaires à l'authentification, puis sur le protocole

d'authentification lui-même.

Remarque : on pourrait utiliser uniquement des authentifiants pour désigner les éléments du réseau. Ainsi, il n'y aurait pas besoin de certifier ni de distribuer l'association entre l'identifiant et l'authentifiant. Par exemple, si l'authentifiant était une clé privée, il suffirait de chiffrer les messages vers un destinataire avec la clé publique de celui-ci (il n'y a aucun problème pour la distribution d'éléments publics). Néanmoins, lorsque les échanges ont pour origine le niveau humain, les identifiants sont absolument nécessaires pour que les utilisateurs puissent reconnaître aisément l'identité des autres individus, sans quoi ils ne pourraient plus cibler leurs communications.

Authentifiants. L'identifiant permet la « reconnaissance » de l'utilisateur et l'authentifiant sert à lier l'utilisateur à cet identifiant. Pour cela, l'identifiant et l'authentifiant sont associés de manière à ce que la connaissance par un individu de ces deux éléments prouve qu'il en est le possesseur. L'authentifiant d'un utilisateur doit donc être caractériser de manière unique. Il peut être de différentes natures :

1. quelque chose que l'utilisateur sait (mot de passe, code PIN, phrase secrète, etc). Il existe de nombreux risques à utiliser des mots de passe : s'ils sont écris on peut les lire, lorsqu'ils sont tapés au clavier quelqu'un peut les voir (on parle de *shoulder surfing* [146]), on peut même les révéler soi-même par imprudence. De plus, si un mot de passe chiffré est intercepté, il peut être soumis à un déchiffrement par dictionnaire ou par force brute [108] (chap 10.2.2).

2. quelque chose que l'utilisateur est. Il s'agit des caractéristiques propres à une personne : empreintes digitales, rétine, voix, etc. Ces éléments sont étudiés dans le domaine de la biométrie [156]. L'utilisation pour l'authentification se développe [81, 107, 129]. Cependant, les techniques de numérisation des données biométriques (pour procéder à des comparaisons) ont été prouvées non fiables à 100% car duplicables [80]. Par exemple, dans [106], les empreintes digitales sont reproduites avec de la gélatine versée dans un moule des empreintes originales (simple pâte à modeler). Onze systèmes biométriques ont ainsi été leurrés à 80%.

3. quelque chose que l'utilisateur a (*token*). On peut se servir d'éléments électroniques capables de stocker les informations d'authentification et/ou de procéder à des calculs sur ceux-ci. Ils doivent être *tamper-resistants* ce qui signifie que les données qu'ils contiennent sont protégées contre des opérations illégales (lectures/écritures non autorisées). Il en existe plusieurs types :
   - les cartes magnétiques ont une faible capacité de mémoire et ne peuvent pas réaliser de calculs. Elles sont très utilisées (certaines cartes de crédit, cartes d'accès, cartes d'abonnement, etc) et très bon marché. En revanche, elles offrent peu de possibilités. Une fois que des informations ont été écrites sur la carte, il n'est plus possible de les modifier.
   - les *smarts cards* [134, 130] possèdent un microprocesseur, de la mémoire et une

(unique) ligne d'entrées/sorties. Cela permet de contrôler les accès aux informations et les calculs effectués (les comparaisons des mots de passe peuvent être réalisées directement sur la carte). Ces cartes sont plus coûteuses. On peut y associer de la biométrie (on parle alors de *bio-token*) ;
- les *hand-helds password generator* sont des dispositifs portables pouvant stocker des clés. Ils permettent en outre d'utiliser des systèmes de type *challenge/response* et des OTP (*One Time Password*, voir paragraphe sur les méthodes d'authentification).

Étant des dispositifs indépendants de la machine principale, une authentification de l'utilisateur est absolument nécessaire. En effet, ces *token* contiennent les informations nécessaires pour une authentification auprès du réseau. L'authentification de l'utilisateur auprès de ce module permet d'apporter la preuve (ou non) qu'il a des droits sur les données stockées.

Il existe aussi ce que l'on appelle des *tokens* logiciels. Les éléments d'authentification ne sont plus gardés ou générés sur un dispositif propriété de l'utilisateur, le *token* est demandé à un serveur (auprès duquel il faut s'authentifier) ou à une application qui va le calculer et l'envoyer au demandeur. Cette approche est décrite dans [74] où sont aussi présentées les différences de sécurité entre les *tokens* logiciels et les *tokens* matériels.

Remarque : il est possible de combiner les solutions. On peut utiliser un *token* auprès duquel on s'authentifie par un système biométrique, puis procèder à une authentification mutuelle cryptographique entre le *token* et le système.

Un exemple d'authentifiant utilisé dans les réseaux est l'adresse MAC (l'identifiant étant l'adresse IP). L'adresse MAC est supposée unique, inscrite définitivement dans le matériel (elle fait le lien entre l'adresse IP et le matériel). Cependant, cette adresse est fournie aux couches logicielles de manière non sécurisée, ce qui permet à un utilisateur de la modifier. Enfin, l'adresse MAC n'est pas secrète, et n'importe qui peut donc prendre l'adresse MAC d'un autre matériel. Il s'agit là d'un problème de sécurité évident, qui donne lieu aux attaques de type « MAC spoofing ».
Des systèmes utilisant les TPM (*Trusted Platform Module*) peuvent être utilisés pour empêcher la modification de ces adresses. Il s'agit d'une puce embarquée dans la carte mère de l'ordinateur qui permet de chiffrer des données et de vérifier qu'elles n'ont pas été compromises par une source externe ou même par l'utilisateur lui-même. Les TPM suivent la spécification TCG (*Trusted Computing Group*, [148]).
Remarque : si l'authentifiant est reproductible il doit absolument être tenu secret.

Distribution des informations d'authentification. L'authentification se fait entre un utilisateur et une autre entité (une autorité reconnue, un autre utilisateur, etc.). Il ne suffit pas que l'utilisateur soit en possession de son authentifiant, il faut aussi que l'autre protagoniste ait un certain nombre d'informations pour vérifier la validité de cet authentifiant. Il existe trois approches selon la nature de l'authentifiant (fig. 19) :

- la méthode utilisant une ACL (*Access Control List*, fig. 19(a)) consiste à fournir à un ou plusieurs serveurs la liste des identités des nœuds autorisés à participer au groupe concerné, ainsi que des informations d'authentification. Si les autorisations d'accès sont gérées dynamiquement (un utilisateur peut devenir membre du groupe n'importe quand), l'administrateur doit mettre à jour toutes les listes à chaque modification de la composition du groupe. Plus il y a d'arrivées de nouveaux membres, plus les interventions de l'administrateur sont nombreuses et plus la tâche est lourde pour le ou les serveurs stockant la liste. De plus, ces derniers doivent rester accessibles aux nœuds utilisateurs ;

- l'authentification peut aussi reposer sur un secret (fig. 19(b), un mot de passe à présenter, une clé, etc.) remis à tous les utilisateurs qui ont été autorisés à entrer dans le groupe concerné par l'autorité compétente. La sécurité des accès au groupe reposant sur ce secret unique, il faut que cet échange soit sécurisé. De plus, il faut protéger ce secret de toute divulgation. Si la confiance accordée aux membres du groupe n'est pas absolue, il faut que les utilisateurs possèdent le secret sans y avoir accès ;

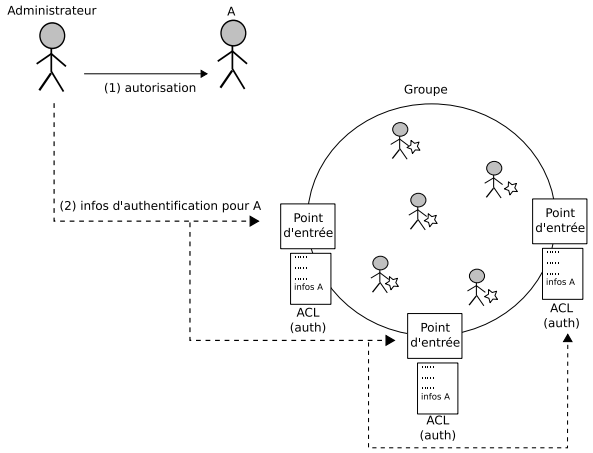
- dans la méthode par certificats (fig. 19(b), chaque nœud possède un certificat personnel signé par l'entité autorisant la participation à un groupe. Tous les membres de ce groupe ont la clé publique de cette autorité pour pouvoir déchiffrer le certificat et vérifier sa validité. Par rapport à l'approche avec secret de groupe, chaque utilisateur est authentifié personnellement donc il serait dangereux pour lui de distribuer volontairement son moyen d'entrée dans le groupe.

Les deux dernières méthodes évitent les mises à jours de listes de participants et ne nécessitent, pour les serveurs, que la connaissance d'un unique élément, à savoir la clé publique de l'entité autorisant les entrées ou le secret de groupe.

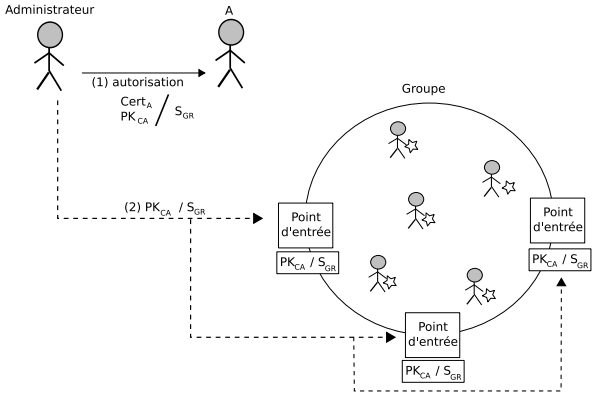
Protocoles d'authentification. Une fois la distribution des informations d'authentification réalisée, il faut procéder à l'authentification à proprement parler. Le candidat, à l'entrée dans un groupe, doit échanger des informations avec un nœud possédant les données nécessaires pour l'authentifier (un serveur d'authentification). Deux méthodes d'authentification à distance sont présentées dans [54], nous les détaillons ci-dessous.

Solution d'authentification faible (fig. 20(a)). Les informations d'authentification (identifiant et mot de passe, secret d'autorisation, etc) transitent sur le réseau. Une interception étant alors possible, les données doivent être chiffrées. L'interception des communications peut permettre de retrouver les informations échangées par une attaque de type dictionnaire ou force brute.

Solution d'authentification forte (fig. 20(b)). Pour éviter les attaques par interception, une authentification forte permet de ne pas faire transiter d'information secrète sur les liens d'interconnexion. On utilise pour cela un système de type *challenge/response* : le serveur envoie un *challenge* toujours différent à l'utilisateur (par exemple un *nonce*, grand nombre aléatoire). L'utilisateur le résout (le signe par exemple) et le renvoie au serveur. Si la réponse est correcte (on peut par exemple vérifier la signature) l'utilisateur est authentifié.



(a) Authentification avec ACL



(b) Authentification avec Certificats ou Secret de groupe

FIG. 19 – Méthodes d'authentification auprès d'un groupe

Cette méthode est sensible aux attaques de type *man-in-the-middle*, rejeu, etc. Pour les éviter, on utilise des mots de passe à usage unique, les OTP (One-Time Password, [108] chap. 10, [75]) : le *challenge* et le mot de passe de l'utilisateur sont passés en paramètres à une fonction de hachage. Le résultat de celle-ci est formaté de manière à constituer un mot de passe en ASCII. Cet OTP n'est utilisable qu'une seule fois. Néanmoins, les méthodes par authentification forte connaissent des faiblesses. Par exemple, l'interception du *challenge* (en clair) et de l'OTP généré peut permettre de retrouver le mot de passe utilisateur.

c) Admission d'un nouveau membre

Une fois le candidat authentifié, le serveur se charge de lui remettre les éléments nécessaires à sa participation au groupe (un certificat, un secret, etc.) et/ou de toutes les mises à jour éventuellement nécessaires (liste des membres admis par exemple). Ceci est en général réalisé par le nœud qui a procédé à l'authentification.

(a) Authentification faible (b) Authentification forte

Fig. 20 – Protocoles d'authentification

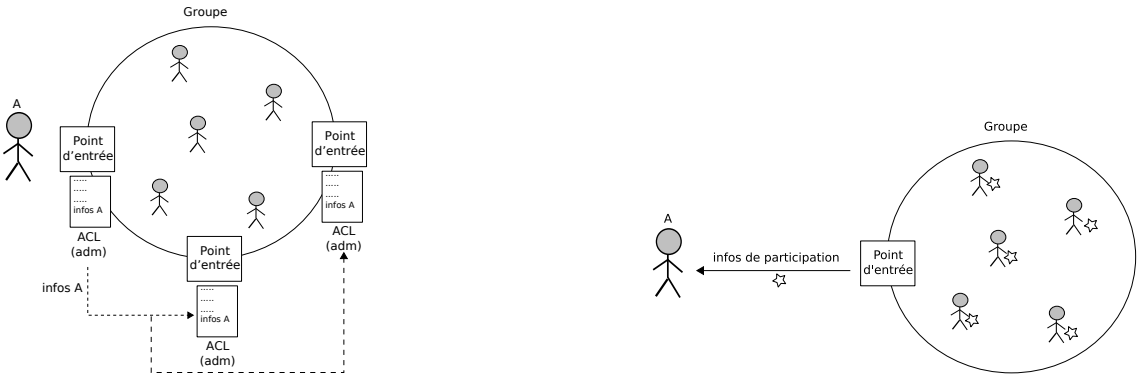
Diffusion de l'information d'admission. Pour qu'un nœud puisse participer à un groupe il faut que son admission soit connue de tous (fig. 21), et donc établir une preuve de cette admission. L'information n'est plus destinée aux serveurs d'authentification mais à tous les membres et les opérations ne sont plus réalisées par ces serveurs mais par les membres eux-mêmes. On peut utiliser le même type de méthodes que celles présentées en 1.4.2(b) (32) pour prouver - non plus que l'on est autorisé à entrer dans le groupe mais - que l'on a été admis. Les méthodes d'admission sont donc les suivantes :

- les identité des nœuds admis sont maintenues dans une liste à la manière des ACL (fig. 21(a)). Il ne s'agit plus d'une liste des nœuds autorisés mais des nœuds admis. Cette liste est ensuite consultée par les (applications) membres du groupe pour vérifier l'appartenance d'un nœud à ce groupe, ainsi éventuellement que ses droits d'accès, avant d'échanger avec lui.
  Si cette liste est dupliquée sur plusieurs serveurs (pour avoir une meilleure disponibilité) et que les admissions sont dynamiques (i.e. les listes peuvent changer), il faut synchroniser toutes les copies de listes (et ce, de manière sécurisée). Si le groupe est de taille importante, la tâche des serveurs peut être lourde et il peut s'avérer délicat d'imposer cette tâche aux utilisateurs. On peut par ailleurs noter que, plus le nombre de serveurs est important, plus l'initialisation est contraignante et plus le nombre d'échanges augmente ;

- un certificat personnel est remis à chaque nouveau membre (fig. 21(b)), signé par une clé globale du groupe concerné et dont tout le monde a la clé de déchiffrement. Le certificat sert directement à participer au groupe et il est remis aux membres de ce groupe par le nœud, ou l'ensemble de nœuds, ayant procédé à l'authentification. Les détails de sa construction et de sa distribution sont étudiés en 1.4.3(b) (page 41) ;

- un secret de groupe est distribué à chaque membre (fig. 21(b)). De la même manière que dans le cas d'utilisation d'un certificat, le secret sert directement à participer au groupe et il est remis par le nœud, ou l'ensemble de nœuds, ayant procédé à l'authentification du nouveau membre. Les détails de sa construction et de sa distribution sont présentés en 1.4.3(c) (page 43).

Authentification et admission par décision collective. Certaines solutions distribuant un secret ou un certificat font coopérer plusieurs membres du groupe [25, 115, 132, 140], afin d'éliminer le besoin de serveurs dédiés à ces tâches. On emploie pour cela, la technique du *threshol secret sharing* présentée en 1.4.1(d) (page 27) : un utilisateur diffuse sa candidature et reçoit avec chaque réponse positive (i.e. si l'authentification a réussie) le



(a) ACL (b) Secret ou certificat

FIG. 21 – Diffusion de l'information d'admission

secret/sa clé publique (pour un certificat) signé par une clé partielle. La reconstitution de la signature complète nécessite la réception d'un nombre n déterminé de réponses positives. Ces solutions présentent plusieurs inconvénients comme la nécessité pour tous les nœuds aptent à voter de connaître les informations d'authentification et par conséquent d'être soumis à une phase d'initialisation. Si ces nœuds doivent stocker la clé publique de tous les nœuds autorisés à participer, le groupe doit être de petite taille.

Des solutions sont basées sur un chiffrement RSA [25, 48, 140]. Le premier schéma a été proposé par Desmedt et Frankel dans [48]. Beaucoup de travaux se sont inspirés de cette méthode. Cependant, la plupart présentent des inconvénients comme le besoin de synchroner des nœuds. Dans les travaux de Shoup [140], la génération des signatures partielles et leur vérification (les membres d'un groupe doivent vérifier que la clé qui leur est distribuée est valide, cf. 1.4.1(d) page 27) sont totalement non-interactives et évitent donc le problème d'inaccessibilité des nœuds (pas de synchronisation nécessaire entre les nœuds).
En revanche, l'inconvénient est qu'il n'est pas possible de repartager les secrets partiels avec un nouvel arrivant dans le groupe. Si on souhaitait une distribution totale, il faudrait donc recalculer entièrement puis redistribuer à tous les membres un nouveau secret à chaque nouvelle arrivée. Ces deux méthodes nécessitent un « dealer » pour générer la paire de clés globale et donc une centralisation à l'initialisation.
Dans la solution [25] la génération de cette paire de clé globale est distribuée pour éviter l'emploi du « dealer ». En contre partie, cette solution ne préserve pas les avantages de la méthode de Shoup [140]. Ces deux solutions gèrent en outre la vérification des clés partielles.

D'autres solutions sont basées sur un chiffrement DSA dans [115, 132] : l'avantage principal de ces solutions par rapport aux précédente est la possibilité de fournir aux nouveaux membres un secret partiel sans tout recalculer. Le principe du « Joint Secret Sharing » [122] (cf. 1.4.1(d) page 26, partie traitant du partage du rôle d'un CA) pour un calcul de la clé globale distribué. Un autre avantage est le seuil (nombre de secret partiel nécessaire pour reconstituer le secret complet) qui s'adapte dynamiquement au nombre de nœuds présents. La vérification des secrets partiels y est prise en compte. En revanche, l'inconvénient est que le nombre de nœuds à corrompre pour retrouver la clé globale est faible.

d) Révocation

La révocation est le mécanisme par lequel les membres d'un groupe sont informés du départ d'un des membres (départ volontaire ou suite à une exclusion). La révocation permet d'empêcher un ancien membre de prétendre faire toujours partie du groupe.

La révocation doit satisfaire trois propriétés :

- lorsqu'un membre a été exclu il ne doit plus être capable ni de prouver son appartenance au groupe ni de participer aux futures décisions d'admission.
- seul un membre du groupe peut prendre part au processus de révocation. Il ne faut pas que les utilisateurs non membres (dont les anciens membres ayant été révoqués) puissent émettre de vote de révocation.
- un groupe de membres exclus ne doit pas pouvoir de collaborer pour retrouver le secret de groupe.

Les révocations peuvent être traitées de deux manières différentes en fonction du mécanisme d'admission employé.

Méthode par liste de révocation. Dans les solutions d'admission reposant sur des ACL ou certificat, on utilise des listes de révocation (CRL) dans lesquelles sont notées au moins l'identité des éléments révoqués. Les problématiques des listes de révocations sont les mêmes que celles des ACL d'admission.

Plus particulièrement pour la révocation des certificats, il existe deux types de système de révocation de type liste :

- les CRL (*Certificate Revocation List* [6, 94]). Avec cette aproche, l'autorité de certification renseigne un serveur de publication qui maintient une CRL. Pour vérifier si un certificat a expiré ou non, un nœud doit faire une requête à ce serveur de publication. Si plusieurs serveurs existent (pour supporter un nombre important de demandes) la synchronisation des listes est une tâche lourde pour l'autorité de certification ;

- CRT (*Certificate Revocation Tree* [114, 118]). Dans cette solution, il n'y a plus qu'un seul serveur de publication sécurisé (qui possède une clé validée), mais il y a plusieurs serveurs auquels on choisit de faire confiance même s'ils ne sont pas sûrs. Cela permet que la clé du serveur ne soit pas répliquée sur plusieurs serveurs ce qui l'exposerait à plus de risques de compromission. Le serveur de publication utilise non plus une liste mais une structure d'arbre pour représenter les informations de révocation. Les feuilles de cet arbre correspondent aux certificats révoqués. La valeur d'un nœud de l'arbre est un haché des valeurs de ses fils et la valeur racine est signée par le serveur de publication sécurisé pour garantir son authenticité. Pour la mise à jour des serveurs de publication, seuls les certificats révoqués sont envoyés. Lorsqu'un utilisateur veut s'assurer de la validité du certificat d'un autre, il interroge le serveur de publication qui lui remet la ou les branches concernées. L'utilisateur vérifie la validité des informations en vérifiant la signature de la valeur de la racine de l'arbre avec la clé publique du serveur de publication sécurisé.
  Un inconvénient de cette méthode est le surcoût dû à la vérification et à la réalisation d'opérations de hachage.
  Remarque : dans certains schémas dits « transparents » (*verifier transparent*), un

serveur intermédiaire est chargé de procéder aux vérification à la place des utilisateurs.

Méthode par rafraîchissement du secret de groupe. Si on utilise un secret de groupe, il faut que celui-ci ne soit plus valable pour un nœud qui a été révoqué. Pour cela, la clé de groupe doit être dérivée et seuls les membres encore autorisés doivent y avoir accès. Cette opération peut être faite à chaque sortie de nœud du groupe ou à intervalles de temps réguliers.
Le problème de la distribution de la nouvelle cle groupe est étudié dans 1.4.3(c) (page 43).

1.4.3 Sécurité des échanges

Une fois entrés dans le groupe, les utilisateurs doivent pouvoir communiquer entre eux de manière sécurisée. Les échanges doivent pouvoir se faire deux à deux ou de un vers tous.

a) Les canaux virtuels

Pour sécuriser les échanges, on met en place des canaux virtuels : les messages sont chiffrés entre deux entités comme les extrémités d'un tunnel. Seules ces extrémités ont accès aux données en clair.

La confidentialité. La confidentialité d'une communication repose sur le chiffrement des paquets transitant d'un interlocuteur à l'autre. La solution la plus utilisée consiste à adopter une combinaison de cryptographie symétrique et asymétrique. On procède en plusieurs étapes :

- génération d'une clé de session ;
- chiffrement de l'information grâce à un algorithme symétrique et à la clé de session ;
- chiffrement de la clé de session à l'aide d'un algorithme asymétrique et de la clé publique du destinataire ;
- envoi de l'information chiffrée et de la clé de session chiffrée.

Les principaux protocoles utilisés sont présentés en [135].

L'intégrité. Assurer l'intégrité d'un message consiste à le protéger de toute modification (illégale) pendant qu'il est échangé entre la source et le destinataire. Pour cela on utilise une fonction de *hachage* qui fabrique un résumé unique du message (appelé empreinte ou *digest*). Le résumé est ensuite joint à l'échange. Le récepteur procède au hachage du message qu'il reçoit et compare le résumé ainsi obtenu au résumé reçu. Si les deux résumés sont parfaitement identiques, c'est que le message n'a pas été modifié.

Authentification. Pour réaliser des communications ciblées, il faut être sûr que l'on s'adresse au bon destinataire. Il s'agit cette fois non plus d'authentifier un utilisateur auprès d'un serveur mais auprès d'un ou plusieurs autres individus pour engager une conversation. Les solutions utilisées sont présentées b) et c).

Non répudiation. La non-répudiation permet de « marquer » toutes les communications de manière à toujours savoir de qui provient l'envoi. Dans les solutions existantes, elle est réalisée en signant chaque échange (chiffrement avec la clé privée). Une réelle non-répudiation des messages dépend de la non-répudiation de la clé utilisée pour la signature. Si un utilisateur peut cacher qu'il a utilisé cette clé, il pourra nier l'envoi du message.

Remarques sur l'interception des échanges. Des précautions peuvent être prises par rapport à l'infrastructure du réseau en charge des communications. Pour les réseaux filaires avec infrastructure, en ce qui concerne la confidentialité et l'intégrité, on peut éviter la diffusion des paquets sur toutes les liaisons physiques par l'utilisation de *switchs* et de câbles sécurisés. Pour les réseaux non-filaires en revanche, ce type de précaution n'est pas possible. Par contre, on peut régler la portée d'émission des dispositifs (même si cette mesure n'est que d'une efficacité toute relative).

b) Gestion des clés pour des communications individuelles

Nous nous plaçons ici dans le cas où des utilisateurs communiquent avec des cibles précises au sein de leur groupe. Chacun possède un certificat associant son identifiant à une clé publique ainsi que la clé privée associée. C'est un schéma classique de PKI (*Public Key Infrastructure*).

Génération et stockage des clés individuelles. La clé individuelle d'un nœud est générée soit par le nœud lui même, soit par un autre membre du groupe (si le nœud n'a pas assez de puissance pour le faire lui-même), soit encore par un serveur de certification (qui peut être le serveur d'admission). Cette clé doit être stockée sur le nœud lui-même ou sur le serveur qui l'a générée.

Authentification des clés individuelles. Il s'agit d'assurer qu'une clé appartient bien à celui qui prétend en être le propriétaire.

Les solutions les plus classiques emploient une autorité de confiance chargée de signer des certificats associant la clé publique de chiffrement et l'identité du possesseur. Ce type de solution contraint les serveurs représentant cette autorité à être disponibles, à réaliser une tâche lourde et éventuellement à stocker l'ensemble des certificats des membres d'un groupe si des comparaisons sur leurs identités sont nécessaires (si l'on veut qu'elles soient uniques) ou si les serveurs font aussi office de KTC (cf. 1.4.1(d) page 26). De plus, des serveurs sont des cibles d'attaques privilégiés et si un seul d'entre eux est compromis, toute la sécurité du réseau l'est aussi. Plus le nombre de serveurs est important (pour une meilleure disponibilité du service), plus les possibilités d'en localiser un et de le compromettre sont grandes.
Une variation de ce type de solution partitionne le réseau en plusieurs domaines, chacun dépendant d'un CA particulier et les CA eux-mêmes dépendent les uns des autres.

Un autre type de solutions consiste à laisser la tâche de certification aux membres d'un groupe, tout en maintenant un bon niveau de sécurité face au risque de l'existance de

nœuds malveillants ou corrompus. Pour cela on utilise la méthode du « threshold secret sharing » présentée en 1.4.1(d) (page 27).

> Les travaux présentés en [96, 97, 102] utilisent ce type de solution en se basant sur RSA. Un « dealer » créé la clé globale et la partage entre les membres initiaux. Dans des recherches basées sur BLS [26, 133], les utilisateurs ont aussi la possibilité de se consulter pour générer la clé globale et la partager entre eux. C'est le cas par exemple dans la solution de Schnoor utilisée en [69].

Dans les solutions de type PGP [147], on élimine toute génération de clé globale. Chaque nœud génère son propre certificat et le fait signer par un nœud qu'il estime être de confiance (ce qui nécessite un système d'établissement de taux de confiance). Un nœud recevant une demande de signature d'un certificat ne le fera que si lui-même pense que le demandeur est honnête. Un nœud accepte ensuite tous les certificats signés par le nœud qui a signé le sien et par ceux dont on a déjà le accepté le certificat. Il se forme alors des chaînes de certification basées sur la confiance.
Ces méthodes sont utilisées dans des contextes particuliers ne nécessitant pas une sécurité très forte car elles reposent sur l'établissement de taux de confiance approximatifs. De plus, des problèmes d'initialisation existent quand peu de nœuds se font confiance au départ.

Dans les solutions de type sous groupe de confiance [72], au lieu d'établir une chaîne de certification, ce sont des sous groupes de certification qui se forment : on établit la confiance au sein d'un groupe de nœuds, puis on recherche des intersections entre les groupes pour créer des chemins de confiance.

Il existe aussi des systèmes de prédistribution de clés [22, 23], dans lesquels, avant de participer à un groupe, chaque nœud pioche individuellement plusieurs clés dans un large ensemble connu. Pour communiquer, deux entités doivent trouver une clé commune qui deviendra leur clé secrète partagée. Cette technique ne permet pas l'authentification d'une entité particulière mais d'une entité ayant le droit de participer à un groupe.

Établissement des clés de session. On considère que l'on dispose déjà d'une clé principale et d'un système d'authentification. Pour communiquer avec un nœud on peut vouloir établir une clé de session c'est-à-dire une clé symétrique utilisée pour une session de communications limitée dans le temps entre deux interlocuteurs. Ceci permet de laisser la clé principale servir uniquement pour l'authentification (pour limiter le risque de corruption) et d'établir des canaux de communication différents pour chaque interlocuteur. De plus, si la clé principale est une clé asymétrique, le chiffrement est beaucoup plus long qu'avec un système à clé symétrique.

Il existe plusieurs types de protocoles d'établissement de clé [108] :
- les protocoles de transport, où seule une des deux entités est chargée de générer la clé et d'en assurer le transport vers l'autre entité. Pour les méthodes asymétriques on parle de *key wrapping* (*Shamir no keys protocol*, *kerberos*, *Needham-Schrœder, otway-rees*, etc.). Pour les méthodes asymétriques on parle de *key pair* (*Needham-Schrœder*, PK, X509, etc.). La plupart de ces protocoles sont présentés dans [108] (chapitre 12).

- les protocoles d'accords, où les deux entités concernées sont requises pour la création de la clé. La clé secrète est dérivée sur la base d'informations propres à chaque entité de manière à ce qu'aucune ne puisse individuellement déterminer la valeur de la clé secrète ainsi construite. Le protocole le plus connu est celui de Diffie-Hellman (station à station) ; il est par exemple utilisé dans [116].

- les protocoles mixtes, où les deux entités sont requises pour la création de la clé et où chacune est chargée du transport des informations nécessaires vers l'autre.

c) Gestion des clés pour des communications orientées groupes

On traite ici des communications qui ont pour cible l'ensemble des éléments d'un groupe.

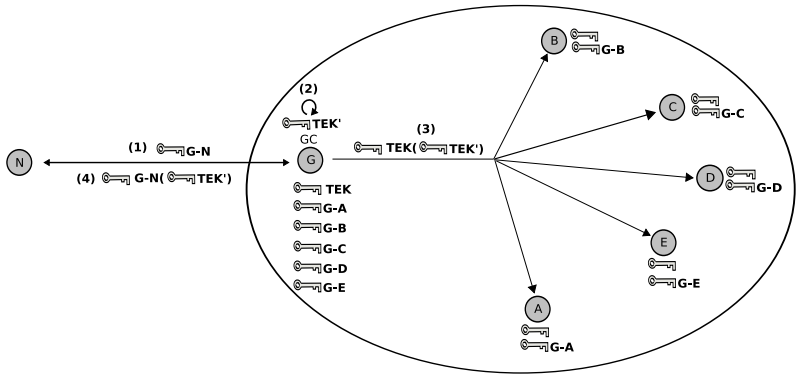
Problématique. La gestion d'une clé de groupe ne s'arrête pas à sa génération et à sa distribution. En effet, si l'entrée dans un groupe est dynamique (de nouveaux membres peuvent arriver et d'autres peuvent partir), il faut pouvoir changer la clé. On parle de *backward secrecy* si la clé de groupe ne permet pas aux nouveaux membres d'accéder aux messages précédant leur entrée, et de *forward secrecy* si les anciens membres d'un groupe (après sortie volontaire, expiration d'un temps autorisé ou exclusion) n'ont plus accès aux messages du groupe après leur sortie.
Nous proposons ici un bref aperçu des solutions existantes, classées en trois catégories : les solutions centralisées où la tâche est réalisée par un nœud central le plus souvent appelé contrôleur de groupe ; les solutions décentralisées où la tâche de ce contrôleur est partagé avec d'autres nœuds du groupe concerné ; les solutions non centralisées ne reposant que sur les membres d'un groupe.

Solution par contrôleur de groupe (GC *Group Controler*). Le principe de cette approche est le suivant (fig. 22) : le GC partage une clé différente avec chaque membre du groupe appelée KEK (*Key Encryption Key*) et une clé commune à tous, la TEK (*Traffic Encryption Key*). A chaque entrée d'un nouveau membre les étapes sont les suivantes (fig. 22(a)) : le nouvel arrivant échange des informations avec le GC (après authentification mutuelle), dont une KEK (1). La clé de groupe est réactualisée par le GC (2). Ce dernier la distribue aux précédents membres en la chiffrant avec la clé de groupe qu'ils connaissent (3), et au nouveau membre par le canal sécurisé dont il dispose avec lui (4).
Au départ d'un nœud (fig. 22(b)) : la clé de groupe est aussi réactualisée (2) mais elle est diffusée par les canaux individuels de chacun (3) afin que le nœud sortant ne puisse pas s'en emparer. On constate de surcroit que cette approche n'est pas très efficace pour les groupes trop larges. On voit, en effet, sur la figure 22(b) que le nombre de messages échangés est très important (θ(n), où n est le nombre de membres d'un groupe). De plus, le GC doit stocker un grand nombre de clé. Ces protocoles ont été introduits par Harney et Muckenhim (GKMP, *Group Key Management Protocol*) [76, 77].

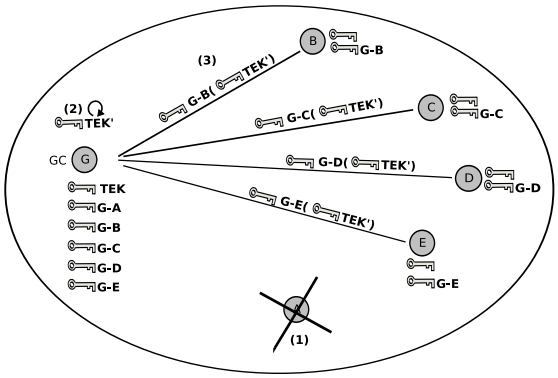
Pour traiter le problème du trop grand nombre de messages diffusés, certaines solutions utilisent une organisation hiérarchique de clés (LKH, *Logical Key Hierarchy*) fonctionnant de la manière suivante (fig. 23) : le contrôleur de groupe créé et maintient un arbre de clés dont les feuilles représentent les membres du groupe concerné. Les nœuds intermédiaires,

eux contiennent des clés et chaque membre possède la clé de tous les nœuds intermédiaires, de la feuille qui lui est associée jusqu'au nœud racine (qui contient la TEK). Pour l'entrée d'un nouveau membre dans le groupe, il n'y a pas de différence avec la méthode précédente. Par contre au départ d'un nœud (fig. 23) les clés de la branche dont le nœud sortant était une feuille sont modifiées par le GC (2) et redistribuées aux feuilles de cette branche (3). Une nouvelle clé TEK est générée (4) mais elle n'est plus signée par la clé de chaque membre mais par la clé du nœud intermédiaire (entre lui et la racine) (5).
Dans ces solutions, moins de messages sont envoyés mais plus l'arbre est profond, plus le nombre de clés à stocker par les membres d'un groupe est sera important.
Les premiers à avoir utilisé ce principe sont Wallner et al. [155] et Wong et al. [157, 158].

> La solution de *Wong et al.* [157, 158] emploie un arbre k-aire alors que celle de Wallner et al. [155] utilise un arbre binaire. L'avantage de la solution de *Wong et al.* est que les membres de groupe ont moins de clés à stocker.
>
> Beaucoup de variations ont suivi : par exemple, dans les travaux de Sherman et MacGrew [138] (OFT, *One-Way Function Tree*), Canneti et al. [35], et Perrig et al. (EFK *Efficient Large group Key distribution*) [123], le nombre de messages échangés pour le protocole de rafraîchissement de clé a été réduit de moitié.



(a) Entrée d'un nœud



(b) Départ d'un nœud

FIG. 22 – Gestion de clé de groupe centralisé (solution simple)

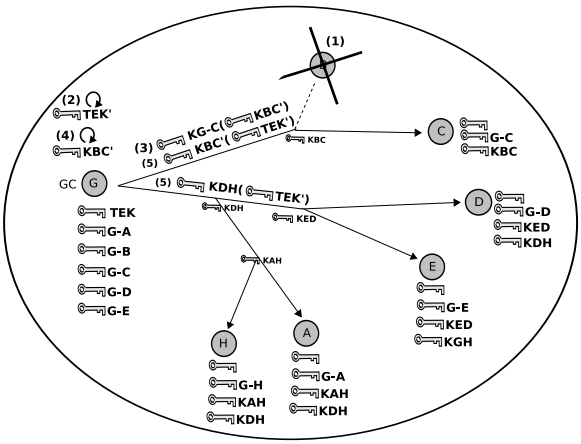


Fig. 23 – Gestion de clés de groupe centralisé (solution hiérarchique)

Solutions par utilisation d'un groupe d'éléments serveurs. La tâche de gestion de la clé de groupe peut être très lourde si la taille du groupe est importante. De plus, le contrôleur est un point sensible aux attaques ou aux pannes. Dans certains travaux, il a été considéré comme préférable de décentraliser au moins une partie de la tâche du GC sur plusieurs nœuds serveurs. Cependant, la plupart des solutions gardent un GC pour générer les clés ou au moins la clé principale TEK et ne résolvent pas le problème de vulnérabilité de ce point central. De plus, toutes ces solutions demandent une préparation importante : plus il y a de nœuds serveurs ou intermédiaires, plus l'initialisation du réseau est lourde.

Dans la solution *Scalable Multicast Key Distribution* (SMKD) de Ballardie et al. [13], un « core » principal [14] créé une ACL, une clé de session (GTEK) et une clé de chiffrement (GKEK) utilisée pour la mise à jour de la clé de session. Ces trois éléments sont distribués à des « core » secondaires et aux autres nœuds au fur et à mesure qu'ils rejoignent l'arbre. Les « core » secondaires peuvent identifier les membres qui souhaitent participer et peuvent leur distribuer les clés. En revanche, seul le « core » principal génère les clés. Cette solution ne propose pas de système de révocation.

Dans Iolus [110], les groupes sont réorganisés en hiérarchie de sous groupes, chacun géré par un agent (CGA, *Group Security Agent*). La particularité est que chaque sous groupe est indépendant c'est-à-dire qu'il possède sa propre clé de communication. Ainsi, si l'un des membres du groupe quitte le réseau, seule la clé du sous groupe dont il fait partie doit être réactualisée. En revanche, on garde une structure hiérarchique puisque chaque groupe est dirigé par un agent GSA (*Group Security Agent*) et un agent principal GSC (*Group Security Controller*) qui gère l'ensemble des GSA.

Les protocoles KHIP [139] (*Keyed Hierarchical Multicast Protocol*) et DEP [51, 50] *Dual Encryption Protocol*) fonctionnent sur le même principe que Iolus mais ils considèrent en outre que les agents ne sont pas forcément sûrs. Le protocole proposé dans [111] gère les problèmes de sécurité générés par l'utilisation de serveurs extérieurs au groupe.

Certaines solutions proposent de faire des rafraîchissement périodiques de clés plutôt que de le faire à chaque entrée ou sortie afin de réduire les coûts associés pour les groupes très dynamiques (Setia et al. [136], Briscoe [31], Yang et al. [99, 161]).

Dans [20] il est proposé d'organiser les membres d'un groupe en différents niveaux de *cluster*, chaque tête de *cluster* pouvant communiquer avec les nœuds qu'elle dirige par *unicast* ou *multicast*. L'inconvénient de cette approche est qu'elle implique beaucoup de calculs et de nombreuses communications.

Solution de gestion par les membres du groupe eux-mêmes. Dans ces solutions, on cherche à éviter l'emploi d'un serveur qui aurait une tâche spécifique et lourde. Les membres du groupe concerné doivent collaborer pour établir et maintenir la clé de groupe.

Beaucoup de ces solutions reposent sur l'algorithme de Diffie-Helman [49] (fig. 24) qui permet l'établissement d'une clé secrète cryptographique entre deux identités.

Principe de l'algorithme de Diffie-Helman : le principe de cet algorithme est qu'à partir de données échangées sur un canal non sécurisé, il est possible de créer une clé secrète entre deux interlocuteurs (fig. 24). La faiblesse de cet algorithme est l'absence d'authentification entre les deux entités ce qui le rend sensible aux attaques de type *man-in-the-middle*.
L'extension de cet algorithme à un groupe demande plusieurs tours (*rounds*) de communication. Tous les nœuds s'identifient entre-eux par un index (numéro d'ordre). Certains d'entre eux joueront un rôle particulier dans le protocole.

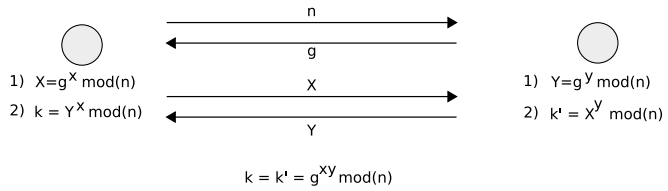


FIG. 24 – Protocole d'établissement d'une clé de session de Diffie-Hellman

Certaines solutions non centralisées ont donc été construites comme des extensions du protocole de Diffie-Hellman [49].

Le protocole de Ingemarsson et al [84] a été le premier à étendre l'algorithme de Diffie-Hellman aux communications de groupes. Dans leur approche, les membres des groupes doivent être organisés en anneau. Ce protocole n'est pas adapté aux mises à jour fréquentes à cause du grand nombre de messages échangés et d'opérations. Il n'est absolument pas adapté aux réseaux dynamiques et mobiles puisqu'il faut connaître l'ensemble des membres du groupe concerné et assurer les communications entre les nœuds dans l'ordre de l'anneau qui les organise.

Le protocole de Steiner et al. (GDH *Group Diffie Hellman*) repose sur le même principe et la même clé que celui d'Ingemarsson. Par contre, il évite la ré-exécution complète du protocole à chaque nouvelle mise-à-jour de la clé de groupe. CLIQUES [144, 145] en est l'extension qui permet de supporter les partitions et les fusions de groupes, ainsi que de très nombreux ajouts et suppressions de membres. En revanche, pour générer une clé de groupe, ces protocoles utilisent des communications deux à deux nécessitant ainsi une exécution en série et donc que les nœuds soient eux-mêmes organisés en série. De plus, le dernier nœud doit savoir qu'il est le dernier ;

Les protocoles Hypercube et Octopus [18], similaires à CLIQUES ont pour but de réduire le nombre de tours ainsi que la taille et le nombre de messages échangés à chaque tour du protocole d'Ingermarsson et al. en organisant les nœuds en Hypercube.

La solution proposée dans [62] repose sur le protocole de Diffie-Hellman. Il y a besoin d'un point central fiable pour initialiser le système et diffuser des informations à tous les éléments ainsi qu'à chaque nouveau membre. En plus du problème de centralisation, ce protocole est vulnérable à la collusion de deux ou plusieurs membres.

Le protocole de Burnester et Desmedv [32] est exécuté en trois tours. Il a les mêmes caractéristiques de sécurité que celles de l'algorithme de Diffie-Hellman. Son exécution théorique pourrait alors être plus rapide mais en fait, chaque tour requiert n *broadcast* simultanés, ce qui n'est (habituellement) pas possible dans les réseaux sans fil. où il ne peut y avoir qu'un seul *broadcast* à la fois. Il faut alors utiliser des *broadcasts* séquentiels ce qui enlève l'avantage du petit nombre de tours. Un autre désavantage est le grand nombre d'opérations.

Boyd [29] a proposé le protocole CKA (*Conference Key Agreement*) qui utilise un système de clés symétriques et des diffusions pour la distribution des informations. Un *leader* du groupe concerné *broadcast* sa contribution cryptée avec les clés publiques à chaque membre de son groupe. Tous les membres de ce groupe peuvent alors calculer la clé de groupe. Ce processus doit être ré-exécuté entièrement à chaque mise à jour du groupe.

Une approche, similaire par l'architecture mais qui n'utilise pas les paramètre de Diffie-Hellman est le protocole PDKD (*Perfectly-Secure Key Distribution for Dynamic Conference*) [24]. Le problème de ce protocole est que la clé de groupe dépend du nombre de membres du groupe concerné. Un point central fiable doit initialiser le protocole.

Le protocole de Poovendra et al. [124] se différencie des autres par la simplicité des calculs nécessaires à l'établissement de la clé de groupe (il s'agit simplement de réaliser un ou exclusif sur des contributions individuelles). Par contre, un initiateur de groupe doit générer et distribuer les paramètres du protocole. Ensuite, les membres s'échangent leur calculs partiels (il faut un canal sécurisé) pour construire le paramètre de groupe. La mise à jour de la clé nécessite une ré-exécution de la deuxième phase.

Ces protocoles sont détaillés dans [40] qui présente les principales solutions de gestion de clés de groupe existantes. Une classification plus complexe et des comparaisons y sont réalisées.

Conclusion

Dans ce chapitre, nous avons déterminé quels étaient les problèmes à prendre en compte pour créer des réseaux sécurisés dans lesquels les membres s'organisent en groupes dans l'objectif de réaliser un but particulier. Nous avons vu que ce problème de la sécurité des réseaux est un sujet de recherche vaste et complexe, mais pour lequel il existe de nombreuses solutions dans des contextes classiques. En revanche, les MANets comme nous l'avons vu constituent un contexte différent et présentent de nouvelles difficultés.

Chapitre 2

Communications sécurisées dans les MANets purs non planifiés

Sommaire

Dans le chapitre précédent nous avons proposé une vue d'ensemble des réseaux et de leur sécurité, puis nous avons plus particulièrement considéré le cas des réseaux MANets. Nous avons ensuite mis en évidence les problématiques liées à la sécurité des communications et nous avons proposé un état de l'art des solutions existant pour sécuriser la partie abstraite des réseaux en fonction de leur contexte.

Dans ce chapitre, nous commençons par présenter les problématiques soulevées par le contexte particulier des MANets purs non planifiés et décrivons l'approche que nous allons mettre en place pour y répondre : un module administrateur local reposant sur l'emploi d'une carte à puce. Dans la deuxième section, nous détaillons le fonctionnement de ce module et décrivons son utilisation pour établir des échanges sécurisés. Enfin, dans les

deux dernières sections, nous utilisons le module administrateur pour sécuriser la gestion des identités et la gestion des groupes.

2.1 La carte à puce : un « administrateur de poche »

Dans cette section nous rappelons nos objectifs et les problématiques associées et en déduisons le principe du module administrateur que nous souhaitons mettre en place. Nous décrivons les cartes à puce qui sont l'outils de base de cette solution. Nous insistons plus particulièrement sur certaines de leurs caractéristiques, importantes à prendre en compte dans le développement d'applications. Nous étudions enfin le cas précis des Java Cards, cartes que nous avons choisi d'utiliser.

2.1.1 Le module administrateur

a) Problématique

Peu des études menées dans le cadre des MANets ont été réellement concrétisées en raison de la complexité des systèmes en l'absence de points centraux et d'administration. Cela libère les utilisateurs dans leurs mouvements mais, en contre partie, ils se retrouvent seuls responsables du bon fonctionnement du réseau. Certaines problématiques deviennent alors extrêmement délicates à traiter (comme la distribution d'éléments secrets par exemple). Actuellement, ces difficultés sont surmontées grâce à une administration ayant pour rôle d'initialiser un certain nombre d'éléments. Elle a pour cela la connaissance de tous les participants au réseau abstrait et ne traite donc que les utilisateurs gérés par une même organisation humaine, celle à laquelle elle appartient.

b) Les MANets purs non planifiés

Nous souhaitons dans ce travail développer une application distribuée ne comptant sur aucune centralisation ni administration, à aucun moment de son cycle de vie. Cela signifie plus particulièrement qu'aucune préparation de nœud par un administrateur ou un quelconque point central n'est possible.
Ainsi, les opérations de contrôle ne peuvent plus être faites sur un serveur mais sur des nœuds utilisateurs. Or tous les nœuds doivent être égaux et capables d'assurer les mêmes tâches afin d'en garantir la disponibilité. En effet, si seuls certains nœuds avaient la responsabilité d'une tâche, il faudrait qu'ils soient accessibles à tous. Or, nous avons vu que ceci représentait une contrainte pour ces nœuds qui n'est pas compatible avec le contexte dans lequel nous nous plaçons. Tous les nœuds doivent donc être capables de réaliser toutes les opérations. De plus, les nœuds étant potentiellement malhonnêtes, deux solutions sont possibles pour réaliser une opération de contrôle :

- l'opération est réalisée par la collaboration de plusieurs nœuds voisins : on espère que sur le nombre, plus de nœuds seront honnêtes que malhonnêtes ;
- si une collaboration n'est pas possible ou pas souhaitée, l'opération est faite par le nœud requérant lui-même, puisqu'il n'est a priori ni plus sûr ni moins sûr qu'un autre.

Pour assurer la sécurité de ces deux solutions, nous proposons dans ce travail d'installer sur chaque nœud un module de contrôle indépendant et protégé de toute intervention de son hôte. Dans la suite de ce chapitre, nous définissons les tâches exactes de ce module.

Dans ce travail, nous souhaitons établir une solution de sécurisation des communications, adaptées à des MANets purs sans aucune planification, en mettant en place sur chaque nœud un module de contrôle sécurisé et indépendant de son hôte.

b) Sécurité du module administrateur

Le module de contrôle que l'on souhaite doit, pour être détenu par chaque utilisateur tout en étant indépendant et protégé de lui, offrir les fonctionnalités suivantes :

- le stockage permanent de données ;
- un fort contrôle d'accès aux données pour les protéger de toute consultation et/ou modification illégale ;
- une capacité de calcul importante (par exemple opérations cryptographiques de création de clés, de chiffrement/déchiffrement) et d'opérations de contrôle diverses sur lesquelles le propriétaire du module ne doit pas pouvoir intervenir.
- la possibilité d'avoir des communications sécurisées entre les différents modules. Ces échanges doivent être transparents et protégés des utilisateurs, ces derniers ne doivent pas y avoir accès et ne doivent pas pouvoir s'interposer.
- la possibilité d'avoir des interactions entre le module et son utilisateur hôte.

Les Java Cards (présentées dans la section suivante) offrent de telles possibilités : les données sont stockées dans la carte à puce et prises en charge par une application nommée *applet*. Seules les méthodes de cette application ont l'autorisation de manipuler ces données. De plus, cette technologie rend (théoriquement) impossible toute lecture du contenu de sa mémoire et toute intervention sur l'exécution d'une application qu'elle embarque. La mémoire persistante des Java Cards permet, en outre, de stocker ces données de manière définitive. Pour ce qui est des opérations complexes, notamment de chiffrement, ces cartes possèdent un co-processeur cryptographique et un générateur de nombres pseudo-aléatoires.
Nous présentons une utilisation des Java Cards pour sécuriser des algorithmes distribués dans [8, 10].

2.1.2 Présentation générale des cartes à puce

Les cartes à puce sont des supports contenant un circuit intégré permettant de stocker des données et de réaliser des calculs. Tout contenu y est sécurisé grâce à des protections matérielles (détaillées en [130]) et un fort contrôle d'accès logiciel. Elles sont petites et légères et sont principalement utilisées pour le stockage et le transport de données secrètes.

a) Types de cartes à puce

Il existe plusieurs catégories de cartes à puce :

- les cartes à mémoire sans processeur (lecture seule sans protection) qui sont non programmables (cartes téléphoniques) ;

- les cartes à logique cablée avec une mémoire préprogrammée et donc capables de réaliser certains calculs spécifiques à une application donnée (décodeurs, carte d'accès à un bâtiment, etc) ;
- les cartes à microprocesseurs, qui possèdent de la mémoire et un processeur. Elles sont programmables et peuvent effectuer des traitements. Ces puces contiennent, en général, un coprocesseur cryptographique pour le chiffrement et le déchiffrement, ainsi qu'un générateur de nombres aléatoires.

C'est ce dernier type de cartes qui nous intéresse dans ce travail, notamment pour leur capacité à effectuer des calculs complexes.

b) Les différentes mémoires

La mémoire est une des caractéristiques essentielles des cartes à puce tant par sa nature que par sa taille :

- la ROM (*Read Only Memory*) est une mémoire « gravée » en usine dont le contenu n'est pas modifiable. Elle permet de stocker le système d'exploitation de la carte et d'autres données permanentes. Elle est, en général, de 32 à 48 kilo-octets ;
- la RAM (*Random Access Memory*) est une mémoire temporaire, volatile, à accès très rapide. Elle perd son contenu dès que la carte n'est plus alimentée. C'est une ressource très restreinte de l'ordre de 1,2 kilo-octets.
- la NVM (*Non Volatil Memory*) est une mémoire persistante accessible en écriture. Il en existe plusieurs types comme l'EEPROM (*Electrical Erasable Programmable Read Only Memory*), la Flash, la FeRAM ou la MRAM. Elle varie de 16 à 64 kilo-octets, mais ceci évolue très vite.

Les cartes à puce sont donc des dispositifs à faible capacité mémoire, une gestion optimale de celle-ci est donc essentielle lors du développement d'applications complexes.

c) La communication carte à puce/terminal

Les communications entre le lecteur et la carte à puce se font en *half-duplex*, c'est-à-dire qu'un seul des deux protagonistes de l'échange peut émettre à la fois. Il faut donc une synchronisation pour ordonner les moments d'émission de chacun. Ceci permet d'éviter la perte de données par collision. C'est le lecteur, par ailleurs source d'alimentation de la carte, qui organise les échanges.

d) Cycle de vie d'une carte à puce

Le cycle de vie des cartes à puce se compose de différentes phases (fig. 25) :

- la fabrication ;
- l'initialisation par l'installation d'éléments logiciels sur l'ensemble des cartes produites (système d'exploitation, outils , etc) ;
- la personnalisation pour une organisation. C'est une phase de différentiation par l'ajout d'éléments spécifiques (par exemple une application ou des informations), en général déterminés par une organisation humaine (comme une banque) ;

- la distribution des cartes, alors restreintes à une application précise, l'organisation pour laquelle elles ont été personnalisées ;
- la personnalisation pour un individu. Les cartes pourront subir un deuxième niveau de différenciation afin de les associer chacune à un membre de l'organisation définie précédemment ;
- la distribution des cartes aux membres du groupe (clients de l'organisation) ;
- l'utilisation par ces membres ;
- la fin de vie des cartes. Les causes de fin de vie peuvent être matérielles, telles que la saturation mémoire, le nombre limité de cycles d'écriture, ou le bris de la carte. Des causes non matérielles peuvent être l'invalidation des informations qui permettent de participer au groupe portées par la carte suite à la révocation de l'utilisateur ou à un vol de cette carte.

Dans les emplois classiques, la personnalisation des cartes à puce limite leur utilisation à une organisation spécifique. A notre connaissance, aucune exploitation de cartes à puce ne permet de rester indépendant de telles organisations.

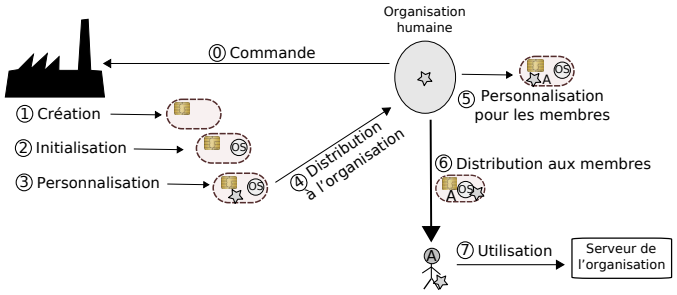


Fig. 25 – Cycle de vie d'une carte à puce

e) Sécurité des cartes à puce.

Les cartes à puce sont sensibles à deux types d'attaques : la consultation des données qu'elles renferment et le clonage.
Ces attaques et les solutions de sécurité existantes sont longuement étudiées dans [130]. Nous en retenons que les cartes à puce sont protégées par de nombreuses mesures matérielles et logicielles visant à empêcher ces deux types d'attaques. C'est pourquoi les cartes à puce représentent un moyen de protection des données extrêmement sûr. Cependant, la sécurité n'est pas absolue et il est important de ne pas perdre ceci de vue lors de la mise en place de solutions les intégrant.

2.1.3 Cas des Java Card

Outre la mémoire et le processeur permettant d'effectuer des calculs et de traiter des informations, les Java Cards ont pour particularité de contenir un ensemble logiciel basé sur la technologie Java :

- l'O.S et des fonctions natives offrant des fonctionnalités bas niveau (entrées/sorties, gestion de la mémoire, co-processeur cryptographique - pas obligatoirement présent sur toutes les cartes -, etc.).

- la machine virtuelle Java Card permettant d'appliquer la politique Java « write once, run everywhere ».
- le *Framework Java Card* (plus de détails seront donnés en section 4.1.4 page 112).
- des classes de base offrant de multiples outils particulièrement utiles pour le développement sur carte à puce.

Les points forts des Java Cards. L'atout principal des Java Cards est l'emploi d'outils Java déjà bien maitrisés par la majorité des programmeurs et ne demandant donc pas de formation supplémentaire. En effet, même si le langage est un sous-ensemble de Java très restreint (cf. 4.1.4 page 112), le développement de codes est hautement facilité par rapport aux langages très bas niveau utilisés jusque là.
De plus, le code produit est universel et n'a pas besoin d'être adapté à chaque type de processeur de carte à puce. Il est particulièrement intéressant pour les programmeurs de pouvoir créer un même *bytecode* exécutable sur n'importe quelle carte.
Un autre avantage provient de l'évolutivité de ces cartes : il est possible d'installer, même après distribution, de nouvelles applications ; ces cartes sont dites multi-applicatives. Du côté client, l'intérêt réside dans la réduction du nombre de cartes possédées.

La sécurité entre applications est parfaitement assurée par un contrôle strict des accès aux données de chacune par un système de *firewall*. Par contre, il est à la charge du programmeur de déterminer la portée de chaque fonction et variable. L'arithmétique de pointeurs n'est pas autorisée afin de rendre impossible les accés à des zones mémoire interdites ou sensibles (cf. 4.1.4 page 112).

2.2 Sécurité des échanges entre modules administrateurs

Les modules doivent communiquer entre eux, les cartes à puce doivent donc échanger par un canal sécurisé accessible uniquement par elles. C'est ce que nous expliquons plus en détail dans un premier temps. Ensuite, nous exposons notre solution de gestion des identités, puis des groupes. Pour chacun de ces trois points nous nous appliquons à mettre en évidence l'utilité des cartes à puce.

2.2.1 Principes et buts

Les données sont protégées tant qu'elles sont à l'intérieur des cartes à puce. Or les tâches que nous souhaitons voir accomplir nécessitent des échanges d'informations secrètes que seul le module administratif (que nous voulons mettre en place sur chaque carte) doit pouvoir consulter. Un canal de communication sécurisé entre les cartes doit donc être construit pour remplir deux missions : la première est de protéger les communications de toute écoute ou modification par des éléments extérieurs ; la deuxième est de prouver qu'un message provient bien d'un module administrateur. Ce dernier point permet d'assurer que les données transportées respectent la politique de sécurité (puisque cela garantit qu'elles sont contrôlées par le module administrateur).

Les échanges entre cartes qui se déroulent par ce canal doivent être transparents pour les utilisateurs. En revanche, un utilisateur doit pouvoir se servir de ce canal pour échanger de

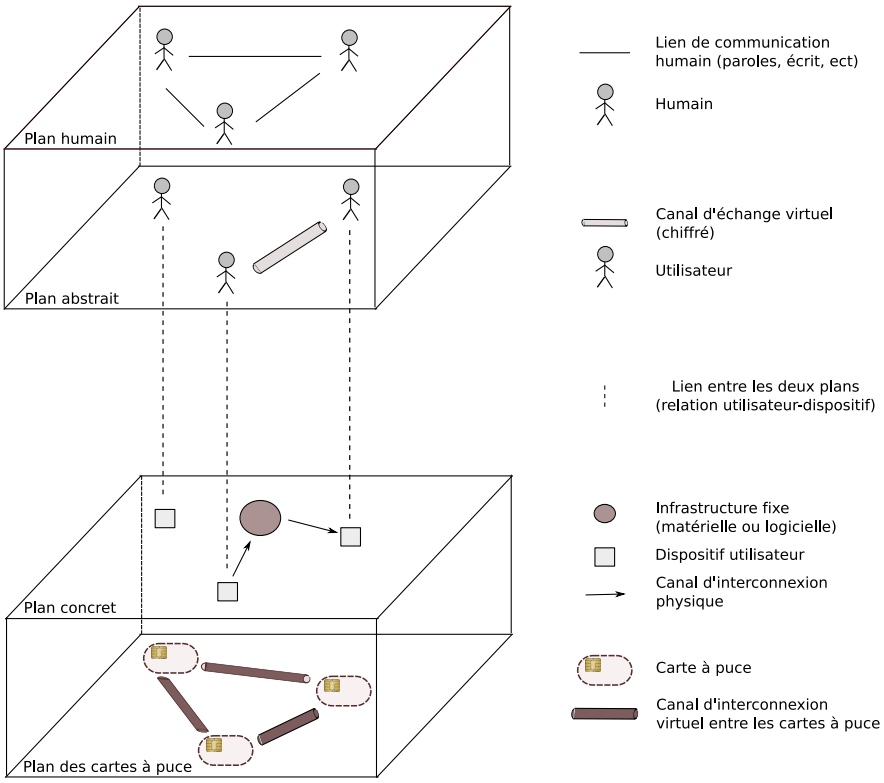


Fig. 26 – Les différents canaux d'un réseau

manière sécurisée des données critiques (clé de session par exemple). Pour cela, l'utilisateur doit pouvoir transmettre le message à sa carte qui doit se charger elle-même de l'envoyer chiffré vers le destinataire. Inversement, du côté du récepteur, le paquet reçu doit transiter par sa carte à puce pour qu'elle le déchiffre puis le retourne à son hôte.

Les différents canaux mis en œuvre dans notre solution sont représentés figure 26.

2.2.2 Construction du canal

Installation d'éléments cryptographiques. Un canal d'échange est dit sécurisé si seules les entités autorisées peuvent manipuler les messages qui y transitent. Le chiffrement des messages contribue notamment à une telle sécurité. Pour ensuite accéder à ces messages, une entité doit posséder soit directement la clé de décryptage, soit des informations permettant de construire cette clé. Dans notre cas, ce sont tous les modules administrateurs qui doivent être en possession des informations cryptographiques nécessaires. Pour que ces informations soient indépendantes de toute organisation spécifique, il faut qu'elles soient installées en usine comme données internes de l'application. Trois politiques s'offrent à nous :

- si une même clé chiffre et déchiffre les échanges, nous sommes dans le cadre d'un système de clés symétriques où toutes les cartes à puce possèdent une même clé secrète globale. Dans les solutions classiques, il n'est pas possible de stocker une telle clé au sein d'un dispositif utilisateur car tout individu aurait accès à cette clé secrète. Or les cartes à puce permettent de préserver ce secret. Toute la sécurité repose alors sur

la fiabilité de la carte. Si, d'une manière ou d'une autre, la carte est compromise et la clé secrète globale découverte, alors l'attaquant pourra créer autant de fausses cartes (virtuelles) qu'il le souhaite.

- si on utilise un système où les cartes stockent cette fois une même paire globale de clés asymétriques (clé privée et clé publique), la problématique est exactement la même que pour la solution précédente à base de clé symétrique.

- si on choisit d'utiliser une paire de clés asymétriques globale mais de ne stocker sur la carte que la clé publique, on ne pourra pas chiffrer les échanges mais simplement déchiffrer des données qui auront été signées par la clé privée (et donc certifiées par l'autorité globale). Nous proposons donc de rajouter une paire de clés asymétriques spécifique à chaque carte ainsi que la clé publique de cette paire signée par la clé privée globale (pour prouver que cette paire provient bien de la carte). Si cet ensemble est découvert, l'utilisateur malveillant pourra cloner à volonté cette carte ou plus précisément produire un système la simulant, le clonage étant une possibilité. En revanche, il ne pourra simuler que celle-ci, puisqu'il n'aura qu'une clé spécifique signée. Dans ce cas, l'utilisateur malveillant pourra par exemple se présenter sous des identités non certifiées (cf. 2.3 page 59) ou participer à des groupes auxquels il n'a normalement pas le droit de contribuer (cf. 2.4 page 66). Il aura aussi la possibilité d'accéder aux conversations de groupes mais pas aux communications privées des autres utilisateurs (voir le chiffrement des données critiques en 2.2.4 page 58). De plus, si un mécanisme permettait de détecter une carte virtuelle (mais ceci est une problématique à part entière qui est hors du cadre de cette thèse) il serait possible de développer un système d'exclusion de cette carte en propageant une information de répudiation à l'ensemble des nœuds du réseau.

Les éléments installés sur une carte ne sont théoriquement pas accessibles de l'extérieur et toutes les solutions proposées pourraient donc être envisagées. Cependant, malgré l'excellente réputation de la sécurité des cartes à puce, elles ne sont pas parfaitement infaillibles (cf. 2.1.2(e) page 53). Le choix le plus sûr est donc la solution dans laquelle on ne stocke pas la clé secrète/privée globale dans la carte.
Le schéma suivant (fig. 27) représente les éléments initiaux se trouvant sur une carte à puce dans notre solution.

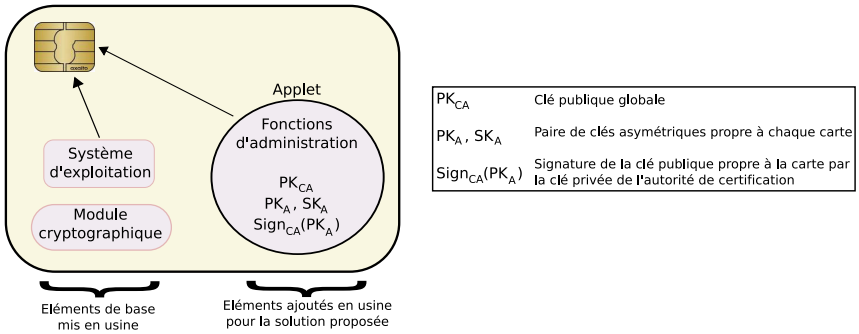


FIG. 27 – Eléments installés en usine sur une carte à puce dans la solution proposée

Discussion sur l'emploi d'un élément global. La notion d'administration globale apparaît ici : nous avons vu qu'elle était obligatoire si on choisissait de ne pas reposer uniquement sur la confiance dans les autres membres. Cependant, dans notre cas, il ne s'agit pas d'une administration classique dédiée à un ensemble fini de groupes particuliers : elle n'a aucune connaissance des éléments du réseau et a pour simple but de fournir les éléments de base comme le module administrateur, la clé publique globale et la paire de clé spécifique. Elle ne gère aucun groupe. Il s'agit d'une étape de préparation supplémentaire du dispositif n'ajoutant aucune contrainte à l'utilisateur et ceci ne constitue pas une préparation du réseau. Cela ne peut donc pas être considéré ni comme une administration du réseau (aucun rôle de gestion), ni comme un point central (les utilisateurs n'ont pas besoin de s'y présenter et/ou d'y laisser des informations les concernant).
De plus, tout en gardant la souplesse d'emploi pour l'utilisateur, cette solution a pour avantage de permettre, en amont, un système de gestion de clés bénéficiant de toutes les solutions existant dans un réseau hiérarchisé tel qu'Internet. Cette approche permettrait aussi par exemple de ne pas utiliser la même clé globale pour signer les clés publiques spécifiques de toutes les cartes à puce produites et diminuerait le risque de corruption de cette clé.

Discussion sur la personnalisation de la carte. La personnalisation de la carte se fait en usine. C'est à ce moment que les paires de clés spécifiques y sont déposées (les différenciant les unes des autres). Ceci est comparable à une mise en œuvre classique des cartes à puce. En revanche, dans notre solution, la personnalisation ne vise aucune exploitation spécifique.

Spécificités de notre utilisation. Les cartes à puce sont aujourd'hui très utilisées pour l'authentification des individus auprès d'une organisation. Beaucoup de vérifications concernant l'utilisateur sont faites au niveau humain et l'identification est en général liée à une identité certifiée (officielle) dans le monde réel. Pour ce qui nous concerne, nous nous plaçons dans un cadre d'échanges entre des utilisateurs et non avec un serveur. Les problématiques ne sont alors plus du tout les mêmes et l'emploi des cartes à puce est très différent de leur usage habituel, comme on peut le voir sur la figure 28.

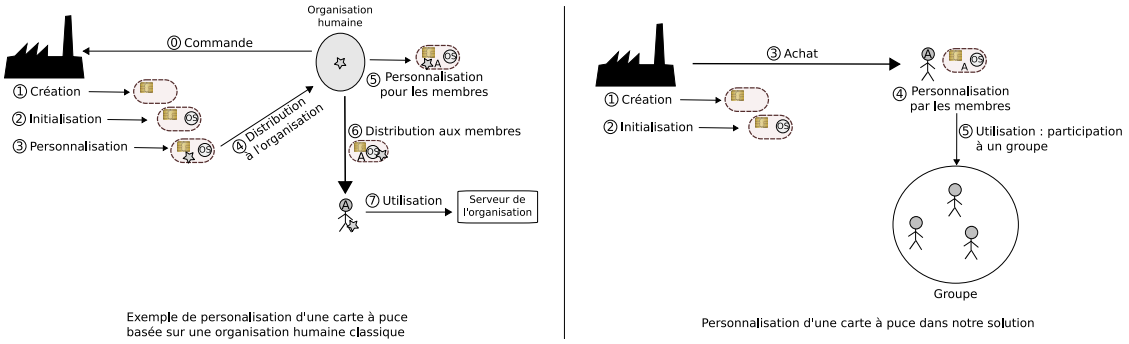


Fig. 28 – Comparaison des étapes de personnalisation entre une application classique et l'application SManet

2.2.3 Sécurité de l'infrastructure des cartes à puce

Peu d'attaques contre le canal de communication entre cartes à puce sont possibles car tous les échanges sont chiffrés. Il n'y a pas de possibilité de *man-in-the-middle* puisque les utilisateurs n'ont aucun contrôle ni aucun accès aux données critiques. De plus, les communications étant chiffrées, il est impossible de savoir à quelle étape du protocole on se trouve. Il est donc difficile de filtrer des paquets en particulier, ce qui permettrait certaines attaques (voir 2.4.3(d) page 76).

Nous verrons par la suite que certaines actions des cartes à puce, suite à une requête, auront pour conséquence le stockage de données. Une attaque par rejeux réémission de paquets enregistrés précédemment) pourrait alors provoquer une saturation de la mémoire de la carte. Pour éviter cela nous associons un *nonce* (grand nombre aléatoire) à toute requête afin qu'il soit repris dans la réponse. Seule une réponse avec un *nonce* valide sera traitée. Ensuite, ce *nonce* sera invalidé pour ne pas gérer plusieurs fois la même réponse (qui proviendrait d'une tentative de rejeu).

2.2.4 Utilisation du canal sécurisé des cartes à puce

Une fois la carte prête, les échanges de données entre les différentes cartes peuvent avoir lieu, chiffrés par une clé stockée dans la carte.

Si les données à transporter sont publiques et que le chiffrement a pour seul but de prouver qu'elles proviennent d'une carte à puce, il doit être fait avec la clé privée spécifique de la carte et le paquet chiffré doit être accompagné du certificat de la clé publique associée (fig. 29(a)) ; c'est un mécanisme de signature classique qui nous permet de prouver que les données échangées proviennent d'une carte à puce.

Si les données à transporter doivent rester secrètes, pour les chiffrer il faut utiliser une clé publique dont la clé privée associée se trouve sur la carte du destinataire (fig. 29(b)). De plus, on choisit de ne pas toujours utiliser la clé publique spécifique des cartes (par exemple, pour l'échange de message entre deux nœuds) pour le chiffrement afin de limiter ses risques d'être compromise. Nous utiliserons plutôt des clés de plus courte durée et d'utilisation moins fondamentale, les clés publiques associées aux identités. Bien sûr, pour prouver que ces messages proviennent d'une carte ils devront aussi être signés par la clé spécifique à la carte.

Parmi les informations chiffrées il y a :

- la nature de la requête (par exemple « Inq » pour une requête de découverte des groupes voisins, les autres types seront décrit dans la suite du document au travers des protocoles) ;
- un *nonce* (un grand nombre) permettant d'éviter les rejeux de requêtes pouvant saturer la mémoire ;
- des données dépendant du type de la requête.

(a) Authentification simple des données

(b) Authentification et confidentialité des données

Fig. 29 – Structure des messages échangés entre cartes à puce

2.3 Gestion des identités par les modules administrateurs

2.3.1 Présentation générale

La gestion des identités permet de supporter la reconnaissance des éléments du réseau. Ceci est nécessaire dans le cadre d'échanges privés afin de cibler les destinataires. En outre, cette reconnaissance est aussi essentielle pour retrouver les nœuds ne respectant pas leurs devoirs et/ou commettant des actes de malveillance.

Dans les MANets, la notion de gestion des identités est en général considérée de manière assez classique : un point central récupère et stocke l'ensemble des identités utilisées dans le réseau. Lorsqu'un nouveau candidat propose une identité, celle-ci est comparée avec toutes celles qui existent déjà. Certains travaux proposent d'utiliser des contextes sans centralisation ni administration (cf. 1.2.5 page 18). Ainsi, dans [91] des clés asymétriques sont choisies comme identité car se sont des éléments uniques par nature (cf. 1.4.2(a), paragraphe traitant de la problématique de l'unicité, page 32). Ensuite, la clé publique est certifiée (ici, signée par la clé privée d'une autorité de confiance), puis installée sur le dispositif (un PDA) en usine. La limite de cette approche est que pour générer une autre identité il faut s'adresser à l'autorité de certification et on retombe donc dans des systèmes administrés classiques. De plus, l'identité peut être effacée, transférée ou volée si elle n'est pas stockée en lieu sûr (ce qui met en péril son unicité).

Au niveau humain, chaque personne est identifiée par un couple (nom - prénom ; apparence physique). Le nom est l'identifiant c'est-à-dire l'élément par lequel on va désigner la personne ; l'apparence physique est l'authentifiant c'est-à-dire l'élément faisant le lien avec la personne et qui la caractérise (quelque chose qu'elle sait, qu'elle a ou qu'elle est, cf. 1.4.2(b) page 33). Dans ce travail, nous adoptons cette décomposition de l'identité.

2.3.2 Politique de sécurité

Nous proposons une politique de gestion des identités permettant de mettre en place une reconnaissance stricte et obligatoire des nœuds. Le but est de n'accepter que les identités respectant certaines règles ; ces règles formeront la politique de sécurité.

Nous avons défini les MANets comme étant des réseaux non dédiés (i.e. où plusieurs groupes indépendants cohabitent, cf. 1.3.1(c) page 20). Ainsi, de la même façon qu'il faut reconnaître strictement les nœuds pour assurer des communications sécurisées, il est nécessaire de strictement différencier les groupes afin qu'il n'y ait pas de confusion possible

entre eux. La gestion des identités que nous proposons doit donc imposer les règles de base suivantes :

- chaque groupe doit être strictement et obligatoirement différencié au sein du réseau (pour qu'on ne puisse pas confondre deux groupes et attribuer à l'un les caractéristiques de l'autre) ;
- chaque utilisateur doit être strictement et obligatoirement différencié au sein de tout groupe et au sein du réseau entier (pour qu'on ne puisse pas confondre deux utilisateurs et attribuer à l'un les communications/actions de l'autre).

Nous ajoutons les règles suivantes :

- une même identité d'utilisateur peut être utilisée dans différents groupes ; en effet, comme on participe à de nombreux groupes, il peut être compliqué et coûteux de devoir générer et stocker une nouvelle identité pour chacun d'entre eux ;
- les individus doivent pouvoir utiliser des identités différentes et choisies par eux dans les différents groupes. Cela permet à un utilisateur qui ne veut pas être « suivi » au travers des groupes auxquels il participe de disposer de plus d'anonymat ;
- un utilisateur ne doit pouvoir utiliser qu'une seule identité pour un même groupe afin qu'il ne puisse pas être malveillant sous une identité et en apparence honnête sous une autre.

Pour que ces règles soient respectées les identités sont soumises à des contraintes. Il faut qu'elles soient :

- uniques et non transmissibles. Ceci permet à toute identité de ne correspondre qu'à un et un seul élément (utilisateur ou groupe). De plus, il faut empêcher les usurpations volontaires (pour commettre des actes malveillants sous l'identité d'un autre dans le cas d'une identité utilisateur / pour tromper les utilisateurs croyant s'adresser aux membres d'un autre groupe, dans le cas d'une identité de groupe) ou accidentelles (mauvaise prise en charge du partitionnement dans un réseau mobile, par exemple) ;

- permanentes. Ceci permet pour les identités d'utilisateurs d'assurer la non-répudiation puisque, pour être sûr de reconnaître un nœud, il est nécessaire que son identité ne soit ni modifiable ni effaçable ;

- associées. On entend par ce terme que pour un groupe, un individu doit obligatoirement se servir d'une et une seule identité d'utilisateur. L'identité du groupe et l'identité d'utilisateur sont alors définitivement associée. Ainsi, aucun participant ne pourra nuire au groupe sous une identité et continuer d'en profiter sous une autre. Si deux identités sont identiques pour des réseaux différents, alors elles désignent le même nœud.

Souhait 1 : les identités des groupes et des utilisateurs doivent être uniques et permanentes. Un utilisateur doit pouvoir avoir des identités identiques ou différentes dans chaque groupe selon son souhait, mais il ne peut en avoir qu'une par groupe.

2.3.3 Problématiques et solution

L'identification des éléments dans les réseaux classiques administrés ne pose pas réellement de problème car c'est un administrateur global qui s'occupe de toute la gestion au niveau humain. Il valide les identités (créées par lui ou par l'utilisateur) et les certifie (fig. 30).

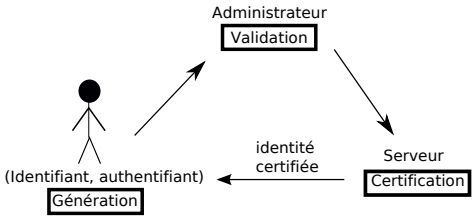


Fig. 30 – Traitement « classique » des étapes de la gestion des identités

Dans notre contexte de travail, sans centralisation ni administration, la gestion des identités se compose aussi de ces trois étapes (création, validation et certification), mais elles doivent toutes être toujours réalisées par le module administratif de chaque nœud (cf. 2.1.1(d) page 50).
Remarque : à ce niveau, il n'est pas possible d'utiliser des méthodes par votes puisqu'il faudrait déjà pour cela avoir des identités sûres.

a) Création de l'identité.

Problématique et existant. Dans notre cadre de travail, les identités sont obligatoirement créées par des nœuds utilisateurs (puisqu'il n'y a pas de serveur). Dans certaines solutions, si un dispositif n'a pas les ressources suffisantes pour réaliser cette opération, il peut demander à un autre nœud de l'effectuer pour lui. Il faut alors pouvoir récupérer cette identité de manière sûre i.e. via un canal de communication sécurisé, ce qui complique le processus, voire le rend impossible. Quoiqu'il en soit, dans notre travail, nous considérons tous les dispositifs comme étant capables de générer leurs propres identités (ce qui pose des problèmes pour assurer l'unicité comme nous le verrons par la suite).

Puisque nous souhaitons établir des communications entre utilisateurs, l'identifiant sera utilisé au niveau humain : au moins une partie de l'identifiant doit donc être lisible (cf. 1.4.2(a) page 31).

Pour qu'une identité soit unique, il suffit que l'un des deux éléments qui la compose (identifiant, authentifiant) soit unique. En général, il est préférable que deux utilisateurs n'aient pas le même identifiant dans un groupe. Nous allons voir dans le paragraphe suivant (validation) qu'une telle unicité ne peut être obtenue que par construction, car aucune comparaison n'est possible avec les identifiants déjà existants.
Il est aussi absolument nécessaire que l'une des deux parties soit secrète : si l'ensemble de l'identité était publique, rien ne pourrait empêcher son usurpation. C'est, en général, l'authentifiant qui reste secret. L'identifiant peut l'être aussi si les communications se font exclusivement avec un serveur (il n'y a pas de communications entre membres donc il n'ont pas besoin de connaître l'identifiant des autres, comme un numéro de compte bancaire

utilisé comme identifiant client, par exemple). Cette partie secrète doit être protégée des autres comme de son détenteur (pour qu'il ne la distribue pas volontairement).

Besoin 1.1 : une identité (d'utilisateur ou de groupe) doit être générée par le module du nœud demandeur lui-même. Elle doit être composée d'une partie publique lisible, d'une partie secrète et d'une partie unique par construction.

Solution proposée. La carte à puce est chargée de générer une paire de clés asymétriques RSA, en théorie unique par nature, qui sera associée au pseudonyme fourni par l'utilisateur. On forme alors la partie identifiant et la partie authentifiant. C'est l'identifiant qui va être distribué au voisinage et c'est donc lui qui doit être unique. Pour cela il est composé du pseudonyme et de la clé publique (unique). Du point de vue utilisateur, si un nœud reçoit deux identifiants avec deux pseudonymes identiques, la carte les modifie par ajout d'un numéro pour les différencier lorsqu'elle les transmet à son utilisateur hôte. L'association du pseudonyme et de la clé publique permet en outre de chiffrer des échanges. Etant non-répudiable, les actes marqués par cette clé le sont aussi.
L'authentifiant est représenté par la clé privée. Pour que celle-ci reste entièrement secrète, aucune méthode ne doit permettre de la présenter hors de la carte.

b) Validation.

Problématique et existant. Une identité doit être contrôlée de manière à s'assurer qu'elle correspond bien à la politique de sécurité choisie.
Nous souhaitons vérifier que les identités créées soient uniques, permanentes et associées (identité d'utilisateur - identité de groupe) :

1. en général, pour assurer l'unicité, un administrateur ou un serveur central compare simplement l'identité proposée à l'ensemble des identités connues qu'il conserve. Dans les MANets purs, l'absence d'administration et de centralisation ne permet plus de réaliser de comparaisons (simples). Nous avons aussi vu en 1.2.4(b) (page 17) que les comparaisons par passage de messages étaient impossibles en raison d'une trop forte « dynamicité » de la topologie et des éventuels partitionnements du réseau en plusieurs sous-réseaux non connexes. La seule solution est alors que l'identité soit unique par construction (cf. 1.4.2(a), paragraphe traitant de la problématique de l'unicité de l'identité page 32).
   La validation de l'unicité consiste alors simplement à vérifier la nature de l'identite créée (si elle est connue pour être unique par construction) et à vérifier que la partie secrète est protégée de toute consultation ou modification, y compris de la part de son détenteur pour en empêcher toute reproduction.
2. pour qu'une identité soit permanente, il faut que personne ne puisse « casser » le lien établi entre l'utilisateur/le dispositif et l'identité. Dans un schéma classique, une association entre l'identité réelle de l'utilisateur et son identité numérique est conservée sur un serveur d'administration. Dans les MANets purs ceci n'est plus possible. L'identité ne peut être stockée que sur des nœuds utilisateurs.
3. l'association définitive d'une identité de groupe et d'une identité utilisateur s'appuie sur une permanence de cette association (même principe que le point précédent) et

l'interdiction pour un individu d'obtenir plusieurs associations impliquant la même identité de groupe.

Besoin 1.2 : pour valider l'unicité d'une identité, chaque module doit pouvoir en vérifier la nature (si elle est unique par construction) puis la protéger de toute reproduction (sinon on perd l'unicité). Pour la permanence, le module doit empêcher l'effacement et la modification des identités. Enfin, pour ne permettre l'utilisation que d'une seule identité par groupe, il faut pouvoir stocker l'association « identité utilisateur - identité de groupe » de manière définitive (ni effaçable, ni modifiable) et unique (pas d'autre association possible avec la même identité de groupe).

Solution proposée. Le module génère lui-même une identité unique par construction et aucune vérification n'est donc nécessaire. Elle reste unique (pas de reproduction possible) car aucune méthode de l'application ne permet d'accéder à sa partie secrète.

La permanance est assurée par le fait qu'aucune méthode de l'application ne permet à un utilisateur d'effacer ou de modifier une identité. Pour garantir l'exclusivité, l'inscription à un groupe est faite par une méthode associant dans un objet les caractéristiques du groupe avec l'identité choisie (cf. 2.4.2 page 66). Cet objet possède un champ « inscrit » que la méthode met à « vrai ». Une nouvelle requête d'inscription sur ce même groupe est refusée par la méthode après vérification de son statut. Il est ainsi impossible de s'inscrire sous plusieurs identités.

Protocole. L'utilisateur fournit d'abord à la carte un identifiant de son choix (1). La carte vérifie que l'identifiant n'est pas déjà enregistré (2). S'il l'est, elle renvoie un message d'erreur à l'utilisateur (6). Sinon elle génère une paire de clé RSA (3), l'associe avec l'identifiant reçu (4) et la stocke définitivement dans la carte (5). Elle renvoie enfin un acquittement à l'utilisateur indiquant que tout s'est bien déroulé (6).

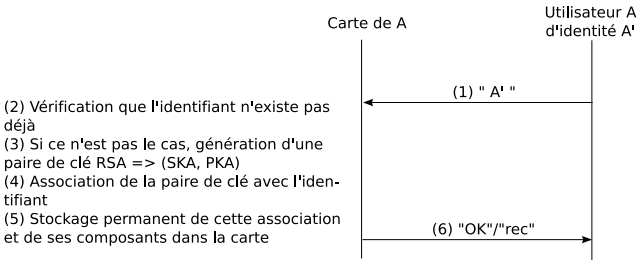


Fig. 31 – Création d'une identité utilisateur dans l'application SManet

Ceci satisfait notre besoin 1.1 puisque le pseudonyme fourni par l'utilisateur représente la partie lisible et la paire de clé RSA la partie unique par construction. La clé privée restera toujours secrète. Nous répondons aussi en partie au besoin 1.2 puisque ces données sont stockées de manière permanente sur la carte à puce (aucune méthode ne permet de

la modifier ou de l'effacer). L'association d'une identité utilisateur avec une identité de groupe s'intègre dans les étapes de création du groupe (ceci sera décrit en 2.4.3 page 69).

c) Certification et diffusion sécurisée.

Problématique et existant. On doit gérer le problème de l'échange des identités entre les utilisateurs du réseau. Il faut que les participants puissent vérifier qu'une identité qui leur est présentée a été validée et qu'elle respecte donc la politique de sécurité précédemment définie.

La certification des identités est une problématique identique à la certification des clés individuelles présentées en 1.4.3(b) (page 41) : il s'agit toujours de prouver que la validité d'un élément a été vérifiée. Nous avons vu qu'il existait plusieurs solutions :

1. la signature par une autorité de confiance ;
2. la signature distribuée par le mécanisme TSS (*Threshold Secret Sharing*, cf. 1.4.1(d) page 27) ;
3. la signature par des nœuds utilisateurs estimés de confiance.

Il est évident que la première solution n'est pas exploitable puisqu'elle repose sur une autorité centrale.

Dans la deuxième solution, la distribution de la clé globale sur un certain nombre de nœuds (ou sur tous), représente une phase d'initialisation lourde. De plus, cette méthode nécessite obligatoirement une organisation humaine commune. Or, nous nous intéressons à une solution qui ne nécessite ni une telle organisation humaine, ni aucune concertation pour une planification.

Enfin, la dernière solution est basée sur un système de confiance qu'il n'est possible d'établir que si les nœuds sont identifiés afin qu'ils puissent s'attribuer entre eux des taux de confiance. Or, aucune identité n'est encore attribuée.

Dans ce travail, si le module administratif créé et valide les identités selon la politique établie, l'assurance que l'identité provient d'un de ces modules constitue une preuve que l'identité a été validée ; c'est donc une certification suffisante (tout ce qui vient d'un module est correct). Les identités doivent simplement transiter entre les modules par un canal accessible par eux seuls. Les modules peuvent ensuite les transmettre à leur hôte.

Besoin 1.3 : pour certifier des identités, les modules doivent pouvoir les échanger entre eux, par un canal de communication sûr qu'ils sont seuls à pouvoir utiliser, ce qui garantie que l'identité échangée provient d'un module (qui est par définition sûr). La problématique se résume donc à celle de la mise en place de canaux sécurisés entre les différents modules.

Solution proposée. Pour répondre à ce besoin nous utilisons le canal sécurisé de cartes à puce mis en place : une méthode distribue l'identité en la chiffrant avec la clé privée spécifique à la carte, accompagnée du certificat de la clé publique associée (cf. 2.2.4 page 58 et protocole ci-après).

Protocole. La validité d'une identité est assurée par l'emploi du canal sécurisé mis en place entre les cartes à puce. Celui-ci est contrôlé par le module de sécurité contenu dans la carte. Si une identité transite par ce canal c'est qu'elle provient d'une carte à puce et donc qu'elle est valide. La distribution se fait au sein d'un groupe. L'utilisateur envoie pour cela une requête à la carte pour qu'elle lance une découverte des voisins avec la partie lisible de l'identifiant *idGroup* du groupe voulu (1). La carte prépare les données (2) contenant l'identifiant du groupe (la clé publique du groupe PK_{gr} et la partie lisible *idGroup*) ajoutée au message de base. La requête est envoyée (chiffrée) au voisinage avec un identifiant de requête $InqG$ (3)(4)(5). Une carte recevant cette requête la déchiffre (6) et, si et seulement si l'utilisateur est membre du groupe et y est actif, sa carte prépare une réponse en ajoutant au message de base son identité publique (clé publique et pseudonyme mais qui, pour simplifier, est représenté ici par $PK_{B'}$) (7). Cette réponse est envoyée chiffrée par la clé publique spécifique de la carte initiatrice (avec un identifiant de requête $RespInqG$). Cette dernière, la reçoit (8)(9)(10) et la déchiffre (11). L'identité reçue est temporairement enregistrée (12) et l'identifiant est remonté au niveau utilisateur (13).

Du point de vue robustesse, la situation est la suivante : si la communication s'interrompt à l'étape (4), il n'y a aucune conséquence (la carte distante restera simplement non découverte) ; si la communication échoue à l'étape (9), il n'y a pas non plus de conséquences puisque personne n'a encore rien stocké définitivement, si bien qu'il n'y a pas de risque de saturation inutile, accidentelle ou malveillante. Encore une fois le voisin membre restera simplement non découvert.

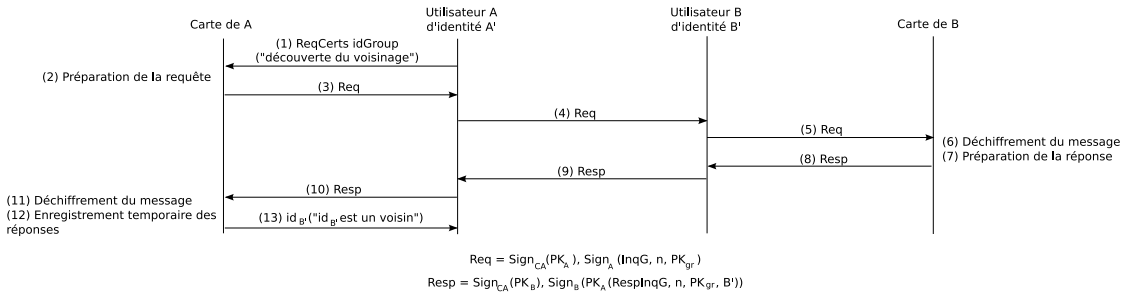


Fig. 32 – Certification et distribution des identités dans l'application SManet

Remarque : il n'est plus nécessaire de stocker des certificats mais simplement les identités.

Conclusion

La politique de gestion des identités que nous souhaitons mettre en place est donc très simple à établir à partir du moment où le canal d'échange sécurisé entre les cartes à puce existe. Ceci nous permet de proposer une solution de gestion des identité gérée en local sur chaque nœud et entièrement sécurisée comme nous le voyons sur la figure 33. Cette partie de notre travail a été présentée dans [9].

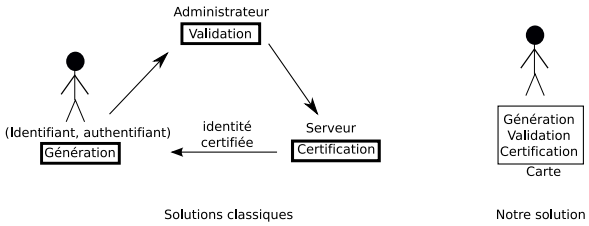


FIG. 33 – Comparaison de la solution de gestion des identités proposée avec les solutions classiques

La gestion des identités permet de différencier les utilisateurs et ainsi de réaliser des communications ciblées et une surveillance des actions de chacun. Pour ensuite permettre des communications sûres il faut étudier les problématiques de la gestion des groupes qui permet de définir le but des échanges et les utilisateurs autorisés à y participer.

2.4 Gestion des groupes par le module administrateur

2.4.1 Présentation générale

Les groupes sont composés de nœuds autorisés à participer à la réalisation d'un objectif commun. Ils sont ainsi caractérisés par deux aspects :

- un aspect concret : leur frontière (notion étudiée en [79, 126]). C'est l'ensemble des nœuds autorisés à en faire partie ;
- un aspect abstrait : le but. C'est ce qui donne la raison d'échanger au sein du groupe et qui rassemble les utilisateurs concernés.

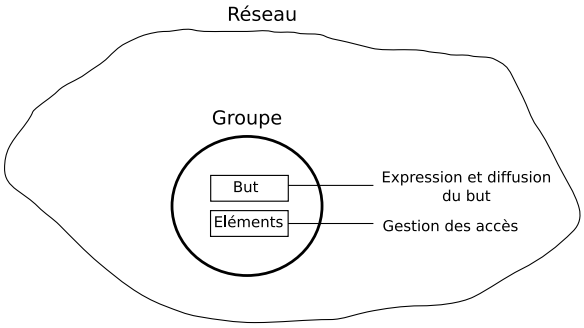


FIG. 34 – Définition d'un groupe

Il faut donc sécuriser ces deux niveaux : pour l'aspect concret il s'agit de mettre en place une politique de gestion des accès ; pour l'aspect abstrait, il faut établir un système d'expression des caractéristiques des groupes (fig. 34).

2.4.2 Politique de sécurité.

Nous considérons deux types de groupe :

1. les groupes à accès ouvert : n'importe quel individu y est accepté, il n'y a aucun critère restrictif d'entrée.

2. les groupes à accès restreint : il existe une condition d'entrée dans le groupe.

En général, les règles d'une politique de gestion des groupes avec intervention d'une organisation humaine sont les suivantes :
- l'autorisation est accordée par l'administrateur de l'organisation humaine ;
- l'admission d'un nœud dans un groupe ouvert ne nécessite aucune condition d'accès ;
- l'admission d'un nœud dans un groupe restreint nécessite la vérification de son autorisation ;
- les admissions sont éventuellement accordées dynamiquement par l'application, c'est-à-dire pendant l'exécution de celle-ci.

Nous souhaitons traiter le cas où aucune organisation humaine n'existe et où aucune interaction directe entre les individus n'est possible. Nous rajoutons donc les règles suivantes (les raisons de ces ajouts seront expliquées plus en détail en 2.4.3(c) page 71) :
- l'autorisation d'admission dans un groupe à accès restreint et l'admission proprement dite dans ce groupe doivent pouvoir être réalisées auprès d'un même nœud qui n'est ni un point central ni l'administrateur d'une organisation humaine ;
- les conditions d'accès à un groupe doivent être gardées secrètes (sinon quiconque peut entrer) ;
- le créateur d'un groupe définit les conditions à satisfaire pour y entrer en fonction du niveau de sécurité qu'il souhaite et des utilisateurs qu'il veut voir participer (ceci permettra la fusion de plusieurs groupes humains au sein d'un groupe unique sans pour autant qu'il soit nécessaire de partager l'ensemble des secrets des groupes).

Souhait 2 : un groupe doit pouvoir choisir son niveau d'accès (ouvert ou restreint). Il faut que les autorisations d'entrée de nouveaux membres puissent être déterminées dynamiquement (i.e. on peut décider d'accepter un nouveau membre qui n'était pas prévu). Les conditions d'accès doivent aussi pouvoir être définies spontanément à la création du groupe. Enfin, la solution doit pouvoir s'utiliser quel que soit le contexte humain (avec ou sans organisation).

Nos besoins combinent deux notions existantes :
- le contrat : ceux qui acceptent les conditions du document s'engagent tacitement à les respecter. Dans le cas contraire, ils sont exclus du groupe ;
- le certificat : il faut assurer que les éléments échangés (identités) sont valides et donc qu'ils proviennent bien d'une carte (celle qui a procédé à tous les contrôles nécessaires).

Nous regroupons ces deux notions dans un unique objet géré par le module administrateur de chaque carte à puce (voir détails ci-après). Nous l'appelons *Objet Personnel de Groupe* (OPG). Il se compose de deux parties : la partie publique (pOPG) contenant toutes les données à présenter à l'utilisateur (identifiant du groupe, description, etc.) et la partie secrète (sOPG) contenant l'ensemble des informations secrètes (réponse au *challenge*, clé secrète du groupe, etc).

Dans la suite, nous présentons en détail la manière dont cet objet est créé, utilisé et échangé entre les utilisateurs.

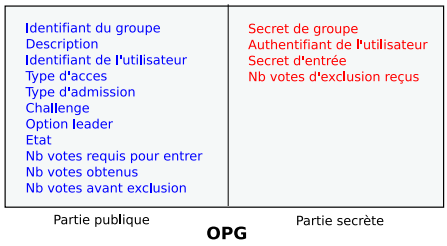


FIG. 35 – Constitution d'un OPG

2.4.3 Problématiques et solution

Nous avons vu dans la section 1.4.2 (page 30) que la formation d'un groupe dans les solutions classiques se faisait en plusieurs étapes. Nous étudions maintenant chacune de ces étapes dans un cadre de MANets purs et déterminons les tâches que devront réaliser les modules (definis en 2.1.1(d) page 50) afin de répondre au souhait énoncé précédemment.

Nous déterminons par la suite quelles tâches concernant la formation des groupes nous souhaitons que le module de contrôle (indépendant et protégé de toute intervention de l'utilisateur) réalise.

a) Création du groupe.

C'est la détermination d'un but particulier, défini par un ou plusieurs individus qui conduit à la création d'un groupe. Ensuite, l'initiateur de ce groupe met en place sa ou ses conditions d'entrée.
Ces deux premières étapes ont obligatoirement lieu au niveau humain et sont réalisées par le créateur du groupe. Elles ne présentent aucun problème particulier puisqu'il s'agit de décisions d'un unique utilisateur qui ne nécessitent aucun échange.

b) Expression des caractéristiques du groupe.

Problématique et existant. Une fois le groupe créé, il faut que les utilisateurs soient informés de son existance et de ses caractéristiques. Ceci leur permettra de juger si l'activité de ce groupe les concerne ou les intéresse. Pour cela, un groupe doit se faire connaître au moins par sa désignation (identité unique) et son but.

Dans les solutions classiques, les caractéristiques d'un groupe sont créées par une organisation qui se charge ensuite de contacter et d'informer ses membres à propos de ces caractéristiques. Par exemple, sur Internet, les groupes peuvent être composés d'utilisateurs n'ayant pas de liens dans le monde humain. Ils se forment autour d'une application centralisée et administrée (« *chats* », « forums », système de ventes aux enchères, de partage de documents, etc). L'expression des caractéristiques de ces groupes est accessible par des systèmes de moteur de recherche ou catalogue (applications centralisées).

Aucune solution d'un autre type n'a été proposée pour les MANets purs sans planification, c'est à dire où aucune organisation et aucun point central ne peuvent informer les utilisateurs. Nous souhaitons donc mettre en place un système permettant de faire la «

publicité » des caractéristiques d'un groupe. Sans centralisation sur un serveur de confiance ce sont les utilisateurs qui sont en charge de « distribuer » ces caractéristiques. Celles-ci doivent alors être protégées de manière à ce que rien ni personne ne les modifie. Pour ne pas surcharger le réseau avec ces échanges, il ne faut pas procéder par inondation à chaque nouvelle création de groupe : ce sont les utilisateurs qui lancent des requêtes de découverte de groupe, et ce, uniquement lorsqu'ils en ont besoin.

Besoin 2.1 : le module doit consigner les caractéristiques d'un groupe dans un document lors de sa création. Il devra ensuite distribuer ce document sur demande, d'utilisateur en utilisateur (en passant par les modules de chacun), sans que ceux-ci puissent en modifier le contenu ou en empêcher la distribution.

Solution proposée. Nous utilisons l'OPG pour stocker les caractéristiques publiques et privées d'un groupe. Le créateur du groupe envoie une requête à la carte et la méthode concernée créée un OPG et remplit ses champs avec les données du groupe. Les caractéristiques sont protégées car nous n'avons implémenté aucune méthode permettant de les consulter ou les modifier. En revanche, une méthode se charge de transmettre à l'utilisateur les éléments publics qu'il doit connaître.

La distribution se fait par le canal de communication entre cartes à puce. Pour le chiffrement, il faut utiliser la clé envoyée par le demandeur (clé publique composant l'identifiant) et non pas la clé spécifique à la carte car les éléments secrets du groupe transitent par le canal (cf. transport des données critiques en 2.2.4 page 58). Tous les échanges étant chiffrés, il n'est pas possible pour un nœud malveillant de reconnaître et donc de bloquer uniquement les demandes de distribution de l'OPG.

Remarque sur la gestion des identités de groupes. Pour différencier les groupes, on utilise le même principe de gestion des identités que pour la reconnaissance des utilisateurs : on génère une paire de clés asymétriques que l'on associe à un pseudonyme de groupe fourni par son créateur. Là encore, l'identité doit être unique pour qu'il ne puisse pas y avoir de confusion entre différents groupes : le clonage des caractéristiques d'un groupe pouvant servir à une attaque dont le but serait de faire croire aux utilisateurs qu'ils échangent avec des membres correspondant à un critère précis. Ainsi, de la même manière que pour les individus, les identifiants de groupes sont constitués d'un pseudonyme et d'une clé publique. Si un nœud découvre deux groupes de même pseudonyme (ce qui est toujours possible puisque ce sont les utilisateurs qui les choisissent), la carte concatènera à ce dernier un numéro pour que les utilisateurs puissent aussi les différencier (ils n'ont accès qu'à ce pseudonyme).

Protocole. Création d'un groupe (fig. 36). Pour commencer, au moins une identité doit avoir été créée (cf. protocole 2.3.3 page 63). Pour créer un groupe, l'utilisateur doit ensuite envoyer un pseudonyme pour le groupe, une des identités utilisateur (qu'il a créées) sous laquelle il souhaite participer au groupe et certaines données de configuration du groupe (mode de vérification, niveau de restriction, type d'organisation, etc) (1). La carte vérifie que le pseudonyme n'existe pas déjà et que le groupe n'est pas déjà créé (2). Si c'est le cas,

elle renvoie une réponse d'erreur à l'utilisateur (5). Sinon elle génère une paire de clés RSA (3) et l'associe au pseudonyme en fournissant à l'OPG toutes les informations (données de configuration évoquées au point (1)) du groupe (4). Enfin, un message d'acquittement est renvoyé au niveau utilisateur (5).

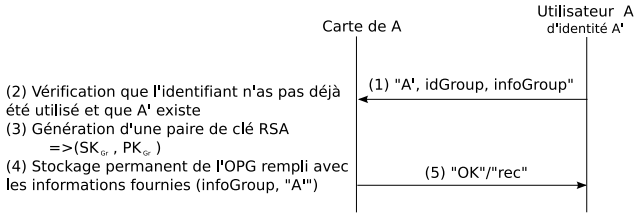


FIG. 36 – Création d'un groupe avec l'application SManet

Ceci termine de répondre au besoin 1.1 concernant l'association définitive entre une identité d'utilisateur et une identité de groupe, dans le cas du créateur du groupe. De plus, l'unicité de l'identité du groupe est assurée puisque sa partie secrète et sa partie unique sont générées dans la carte.

Découverte d'un groupe (fig. 37). L'utilisateur envoie une requête à sa carte pour qu'elle lance une découverte des groupes voisins (1). La carte prépare une requête avec l'identifiant de requête *Inq* (2) contenant seulement le type d'opération et un *nonce*. Cette requête est envoyée (chiffrée) au voisinage (3)(4)(5). Une carte recevant la requête la déchiffre (6) et prépare une réponse en ajoutant l'OPG au message de base (7). Cette réponse (qui a pour identifiant de requête *RespInq*) est chiffrée avec la clé publique du demandeur pour que les données sensibles de l'OPG (secret d'entrée par exemple) ne soient pas lisibles. La réponse est ainsi renvoyée (8)(9) à la carte initiale (10) qui la déchiffre (11). La réponse est temporairement enregistrée (12) et les caractéristiques publiques du groupe sont remontées au niveau utilisateur (13).

Du point de vue de la robustesse la situation est la suivante : si la communication s'interrompt à l'étape (4) il n'y a aucune conséquence (la carte distante restera simplement non découverte) ; si la communication échoue à l'étape (9), il n'y a pas non plus de conséquences puisqu'aucun OPG n'a été stocké définitivement, ce qui ne peut donc pas constituer un risque de saturation inutile, accidentelle ou malveillante. Encore une fois, les groupes resteront simplement non découverts.

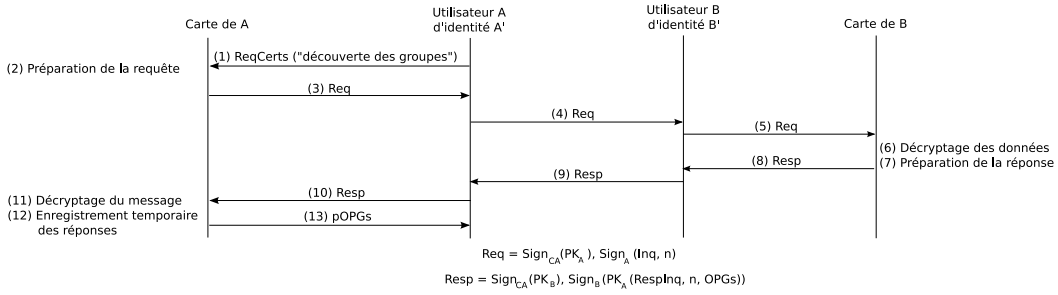


FIG. 37 – Découverte d'un groupe dans l'application SManet

c) Gestion des accès à un groupe

La gestion des accès à un groupe a pour rôle d'assurer que tous ceux, et seuls ceux qui correspondent aux conditions d'entrée de ce groupe puissent y être admis. De cette manière, on protège le but du groupe et l'accès aux informations qu'il véhicule. Cette gestion des accès repose sur plusieurs étapes que nous détaillons maintenant

Problématique et existant concernant l'autorisation. Pour les réseaux ouverts tout membre est accepté et aucune intervention humaine n'est nécessaire (c'est le cas de beaucoup de groupes sur Internet). Pour les groupes à accès restreint, l'entrée est soumise à condition et nécessite l'obtention d'une autorisation. Le critère d'autorisation d'entrée est décidé au niveau humain et peut être de nature très variée : identité réelle d'un utilisateur, aptitude (capacité de calcul, possession d'une voiture rouge, etc), connaissance, etc.

Dans les schémas classiques, l'administrateur d'une organisation humaine est chargé de déterminer qui a le droit de devenir membre du groupe ou non. C'est aussi lui qui transmet l'information d'autorisation au groupe et/ou remet à l'utilisateur la preuve qu'il a été autorisé à participer à ce groupe (voir le paragraphe suivant sur l'authentification). Aucune solution proposant une autre stratégie n'existe pour les MANets purs sans planification. Nous souhaitons ici que la demande d'autorisation soit réalisable auprès d'un membre quelconque du groupe. Tous les membres doivent donc avoir la capacité de juger si un utilisateur peut être autorisé à entrer dans le groupe concerné.

Besoin 2.2 : la ou les conditions d'entrée dans un groupe doivent être remises au module par le créateur de ce groupe (car c'est le module qui est en charge de tous les contrôles dont l'authentification et les vérifications nécessaires à l'admission de membres dans le groupe). Ces éléments doivent être distribués d'utilisateur en utilisateur sans que ceux-ci puissent les modifier ni y accéder (pour la partie secrète). Ce mécanisme est le même que celui évoqué dans le besoin 1.3

Problématique et existant concernant l'authentification. Nous rappelons que l'authentification auprès d'un groupe consiste à apporter la preuve que l'on a été autorisé à entrer dans le groupe. Pour qu'un nœud puisse procéder à l'authentification d'un candidat, il doit posséder certaines informations secrètes d'authentification (cf. 1.4.2(b) page 32). La problématique est la distribution de ces éléments aux dispositifs en charge de l'authentification.

Les solutions classiques (présentées en section 1.4.2(b), paragraphe « distribution des informations d'authentification » page 34) utilisent soit des listes de participants autorisés, soit des certificats (document unique pour chaque nœud signé par une autorité de confiance), soit des secrets de groupe. Aucune des solutions ne convient parfaitement au contexte dans lequel nous travaillons pour les raisons suivantes :

1. Dans les MANets, une méthode par liste de participants autorisés obligerait certains ou tous les nœuds à stocker cette liste. De plus, si la liste est dynamique (ajout d'utilisateurs pendant la vie du groupe), il faut la réactualiser régulièrement sur tous les nœuds qui la gérent. Si les autorisations sont données par une autorité extérieure, il

faut que celle-ci intervienne régulièrement pour une telle réactualisation.La contrainte de stockage et de mise à jour des listes est trop lourde pour les MANets.

2. Les méthodes par certificats signés par une autorité de confiance nécessitent que les nœuds procédant à l'authentification possèdent la clé publique de l'autorité de confiance. La préparation des nœuds, telle qu'elle est proposée dans ce type de solution, est une contrainte que nous souhaitons éviter.

3. Les méthodes par secret de groupe présentent les mêmes problématiques que par certificats.
   Remarque : on rappelle que la seule différence avec l'approche par certificat est qu'un secret n'est pas lié à un unique individu (contrairement aux certificats) et il peut donc être transmis volontairement par les détenteurs à des membres non autorisés.

Dans les MANets purs sans planification, aucune de ces solutions ne convient. La distribution d'éléments d'authentification n'étant pas possible, ce sont les membres du groupe qui doivent directement être interrogés pour décider d'autoriser un nœud à participer à ses activités ou non.

Remarque : une fois le problème de la distribution des éléments d'authentification résolu, il reste à choisir un protocole d'échange des informations entre le candidat et un nœud membre pouvant procéder à l'authentification. Ce sont les protocoles existants pour les réseaux classiques qui sont utilisés.

Besoin 2.3 : si le module possède les informations secrètes explicites (preuves d'autorisation) permettant l'authentification auprès du groupe (en général c'est le cas avec une organisation humaine sous-jacente), il doit se charger de procéder directement aux vérifications nécessaires lorsqu'il reçoit une requête de demande d'autorisation.
Si le module n'a aucune information concernant les « réponses » autorisant l'admission (parce qu'il n'y a pas d'organisation humaine pour les distribuer), nous souhaitons le rendre capable d'utiliser et de gérer un vote engageant les membres voisins (puisque le module n'a pas de réponse précise, les membres sont les seuls à pouvoir juger des réponses proposées), de protéger les réponses à ce vote (contre toute consultation, modification ou interception) puis de les traiter (calcul du nombre de votes positifs, protection de cette valeur contre toute modification, etc.). Le nombre de votes nécessaire à l'admission doit être décidé à la création du groupe par son fondateur.

Problématique et existant concernant l'admission. Il faut ensuite prouver que l'authentification a réussi.

Les méthodes existant pour prouver une admission sont les mêmes que celles pour prouver une autorisation : par liste, par certificat ou par secret commun. La plus grande partie des solutions existant utilise des méthodes de certificats signés par une autorité de confiance avec distribution de la clé sur plusieurs nœuds. D'autres solutions utilisent un secret de groupe. Ces deux types de solution sont détaillées dans la section 1.4.2(c) (page

36). Dans tous les cas, tout comme pour la preuve d'autorisation d'entrée dans le groupe ces méthodes ne conviennent pas aux MANets sans planification car elles nécessitent au moins une phase d'initialisation.

Des mécanismes d'admission ont été étudiés spécifiquement pour les réseaux MANets [95, 115, 131, 132], mais toutes les solutions proposées reposent sur un point central. Par exemple, dans [95, 131], une autorité appelée *Group Authority* (GAUTH) est capable de délivrer des *Membership Certificates* (GMC). Les conditions d'admission sont consignées dans une
« charte de groupe » préparée à l'avance par l'organisateur du réseau. L'admission peut être réalisée par un administrateur et/ou ses délégués, ou par une collectivité (les membres des groupes eux-mêmes par un système de vote)[1]. Ces notions sont présentées dans [95]. Nous avons plus particulièrement décrit les techniques de vote en 1.4.2(c) (paragraphe « Authentification et admission par décision collective », page 37) ; dans [115, 132], ils sont mise en place spécifiquement dans des réseaux MANets.

Besoin 2.4 : le module doit s'occuper de générer une preuve d'admission, de la stoker et d'empêcher tout accès extérieur à celle-ci. La certification de cette preuve d'admission (preuve que cet élément a été délivré par une entité de confiance) repose sur le fait qu'elle n'est échangée que par un canal accessible uniquement par les modules (voir besoin 1.3).

Solution proposée pour les besoins 2.3 et 2.4. Pour devenir membre d'un groupe il faut tout d'abord récupérer les données de l'OPG du groupe par une requête auprès d'un membre (voir le protocole de la découverte des groupes page 70). L'utilisateur fait ensuite une demande d'entrée dans le groupe auprès de sa propre carte en lui envoyant les éventuelles informations nécessaires pour être admis. La carte consulte les conditions d'entrée dans le groupe et plusieurs scénarios sont possibles :

- pour un groupe à accès ouvert, la carte accepte automatiquement l'utilisateur et l'inscrit.

- pour un groupe à accès restreint avec organisation, le créateur du groupe fournit à l'OPG le secret commun (les autorisations sont décidées par l'organisation qui remet le secret aux membres). Le candidat récupère l'OPG et demande à la carte l'inscription à ce groupe en lui fournissant sa réponse au *challenge* (élément public transmis au niveau utilisateur). La méthode correspondante procède directement aux vérifications (comparaison avec le secret entré par le créateur). Si la réponse est validée, la méthode passe l'état de l'OPG à inscrit.
  Plusieurs *challenges* peuvent être associé dans un OPG pour créer un groupe informatique composé de plusieurs groupes humains organisés. Les secrets sont gardés dans l'OPG sans jamais être révélés au niveau humain (les comparaisons sont faites sur la carte, aucune information n'en sort).

[1] Nous verrons dans ce document que nous utiliserons ce principe de vote pour l'admission d'utilisateurs dans un groupe.

- pour un groupe à accès restreint sans organisation humaine (pas de secret commun) l'autorisation et l'admission se font en une seule étape. Le créateur du groupe fournit une condition d'accès à l'OPG. Lorsque le candidat fournit des informations à la carte, deux cas de figure existent :
  - la réponse doit être validée par les membres voisins. Le candidat propose une réponse à l'ensemble des voisins membres. Celle-ci est transmise aux utilisateurs membres (par le biais de leur carte à puce) et ceux-ci jugent eux-mêmes de sa validité. Si le candidat obtient un nombre de réponses positives suffisant (nombre fixé par le créateur du groupe) d'individus différents (ce qui est vérifiable grâce à la gestion des identités mise en place), la carte procède à son inscription. La vérification de la condition dépend de la capacité de jugement des utilisateurs.

  - la réponse doit être validée par un module fourni dans l'OPG. La méthode fait alors passer les informations nécessaires (une réponse de l'utilisateur, le taux d'énergie du dispositif, ses capacités de calculs, sa marque, etc) à ce module qui se charge directement de valider ou non la réponse.

Une fois authentifié, pour prouver que l'on appartient bien à un groupe, il suffit, grâce à l'emploi de la carte à puce, de vérifier que l'OPG a bien été enregistré et activé.

Remarque 1, a propos de la solution d'autorisation et d'admission par vote : si le nombre de membres du groupe est inférieur au seuil fixé pour pouvoir entrer, une autre méthode doit être adoptée en attendant que le groupe soit assez dense. Par exemple, pour notre solution, nous avons choisi, dans un premier temps, d'utiliser un moyen simple : les membres recevant la demande d'entrée dans le groupe renvoient, avec leur jugement, le nombre de leurs voisins. Le candidat fait la moyenne de ce nombre qui deviendra le seuil de cette première phase.

Remarque 2, a propos de la solution d'authentification directe : il serait intéressant que le créateur d'un groupe puisse choisir des conditions d'entrées dépendant de la réalisation d'opérations particulières mais non forcément sur toutes les cartes à puce. Pour cela, il faudra que du code puisse être transporté par l'OPG, puis installé sur les cartes à partir de l'intérieur même de ces cartes. Une telle solution présente de nombreuses problématiques (s'assurer du caractère non malicieux du code, impossibilité de modifier le code, vérifier que les autres membres exécutent le bon code, etc.) qui pourraient faire l'objet d'une étude spécifique.

Remarque 3 : nous avons ajouté à notre solution la possibilité pour un membre d'être actif ou non dans le groupe, c'est-à-dire de participer ou non aux communications, d'être visible ou non. Ainsi, l'utilisateur peut ne pas participer à tous les groupes en même temps. C'est une méthode de la carte à puce qui passe l'OPG en mode actif ou inactif sur demande de l'utilisateur. L'intérêt est d'économiser de l'énergie.

Protocole. Connexion à un groupe (fig. 38, 39). La connexion à un groupe peut se faire selon trois cas de figures en fonction de la configuration choisie par le créateur.
Dans le premier cas de figure (inscription automatique), il suffit à l'utilisateur d'envoyer à la

carte l'identité du groupe, l'identité sous laquelle il souhaite y participer et, éventuellement, une réponse au *challenge* associé au groupe considéré (1). La carte vérifie alors que le groupe n'est pas déjà enregistré, que l'identité associée est bien valide et, que s'il y a un *challenge*, la réponse fournie est correcte (2). Si une de ces conditions n'est pas remplie, la carte renvoie un message de refus à l'utilisateur (4). Sinon, elle inscrit celui-ci au groupe en remplissant et stockant un OPG correspondant (3). Elle renvoie ensuite à l'utilisateur un message de confirmation (4).

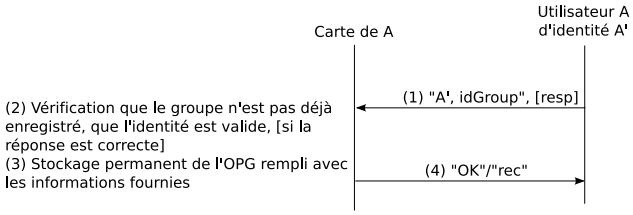


Fig. 38 – Admission automatique dans un groupe dans l'application SManet

Dans le deuxième cas, c'est-à-dire pour une admission par décision humaine, la vérification de la réponse au *challenge* pour participer au groupe doit être faite par un certain nombre de membres du groupe (fig. 39). Ainsi, après l'étape (1), outre les vérifications sur le groupe et l'identité de l'utilisateur, la carte prépare une requête à envoyer au voisinage dans laquelle elle entre la réponse de l'utilisateur au *challenge*, l'identité du groupe désiré et l'identifiant de requête *ReqConn* (2). Une carte voisine, appartenant à un membre du groupe, reçoit la requête (3)(4)(5), la déchiffre (6) et transmet les données reçues (réponse au *challenge*) à son utilisateur (7). Celui-ci prend sa décision et remet son vote à la carte (9) ; celle-ci prépare la réponse (identifiant de requête *RespConn*) en la chiffrant avec la clé publique spécifique à la carte initiale (10) et lui envoie (11)(12). La carte reçoit cette réponse (13)(14) ainsi que celle de toutes les autres cartes des membres voisins. Lorsqu'elle reçoit suffisamment de votes positifs, elle inscrit l'utilisateur au groupe en enregistrant l'OPG (15). Elle renvoie alors un message informant l'utilisateur de son inscription (16).

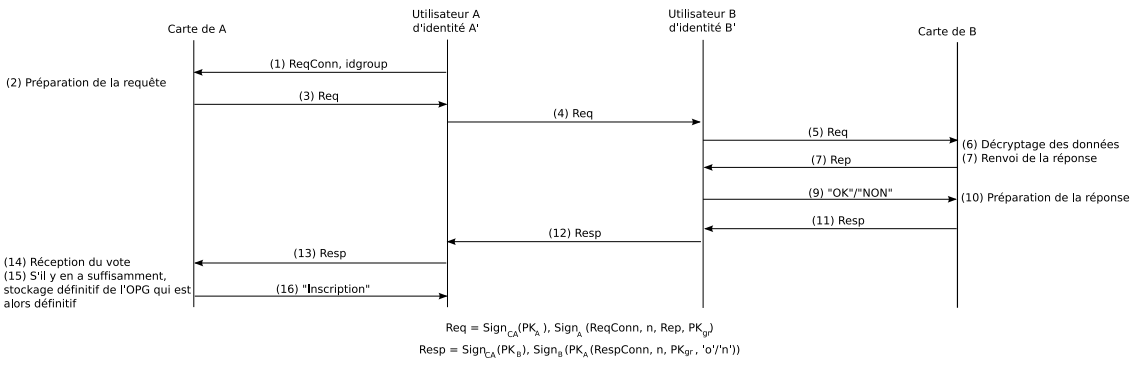


Fig. 39 – Admission par vote dans un groupe dans la solution SManet

d) Révocations.

Problématique et existant. Dans un groupe il est nécessaire de pouvoir éliminer des nœuds jugés malveillants. Les solutions pour réaliser cette fonction dépendent du mode d'admission choisi.

Les solutions classiques proposent principalement des listes de révocation de certificats dans le cas où la preuve d'admission est un certificat, ou des renouvellements de secret (clé secrète) dans le cas où la preuve d'admission est un secret de groupe.

Nous avons vu précédemment que les mécanismes par certificats ne convennaient pas aux MANets purs sans planification. C'est pourquoi les travaux existant se sont plutôt penchés sur les méthodes de dérivation d'une clé de groupe (remise comme preuve d'admission) : un secret de groupe est valable à un moment donné pour un ensemble de membres donné. Or cet ensemble évolue au cours du temps puisque des nœuds entrent et d'autres sortent. Les nœuds qui sortent (exclusion ou temps de participation autorisé écoulé, par exemple) ne doivent plus avoir accès au groupe. Nous avons vu en 1.4.3(c) (page 43) que les solutions existant pour les réseaux dynamiques se basaient sur une réactualisation des clés à chaque sortie ou par intervalles de temps. Ces méthodes ont des phases d'initialisation importantes et reposent sur des contrôleurs, points centraux ayant une tâche plus importante que les autres nœuds. Il faut, en outre, à chaque réactualisation, redistribuer la nouvelle clé à tous les nœuds encore membres. Ce type de solution est trop lourd dans notre contexte.

En ce qui concerne les décisionnaires de la révocation d'un nœud, beaucoup de solutions ont mis en place un système de vote contre un voisin s'ils jugent qu'il a un mauvais comportement. Dans ce travail, nous souhaitons pouvoir bannir des utilisateurs qui ne respectent pas le but du groupe auquel ils appartiennent. Il s'agit donc de juger des actes visibles par les utilisateurs (si le but du groupe est de réaliser une tâche, les voisins peuvent juger du résultat). Le nombre de votes avant exclusion doit être décidé à la création du groupe par son fondateur. Ensuite, il faut qu'un utilisateur visé par des votes d'exclusion ne puisse pas s'interposer et empêcher leur traitement, ni les modifier.

Besoin 2.5 : pour la révocation d'un membre d'un groupe, le module doit reprendre le même mécanisme de vote que pour le besoin 2.4.

Solution proposée. Nous avons donc implémenté une méthode qui reçoit les votes négatifs des voisins et les comptabilise. Elle ne considère qu'une seule fois le vote d'un même membre en gardant les informations nécessaires. Ces dernières sont inaccessibles depuis l'extérieur (aucune méthode ne permet d'intervention non autorisée sur ces données). Les seuls votes d'exclusion ne peuvent pas être interceptés et écartés car toutes les communications entre cartes à puce sont chiffrées. Lorsque le nombre de votes d'exclusion atteint le seuil fixé par son créateur, une méthode passe le statut de l'OPG à exclu. L'OPG caractérise le groupe entier et ne fait pas de différence entre les îlots. L'exclusion est donc valable dans tous les sous-groupes non connexes. Dans notre solution actuelle, l'utilisateur peut réitérer le processus d'accès au groupe.

Protocole. Pour la révocation, il suffit à un utilisateur d'envoyer un vote d'exclusion. Il s'agit d'une donnée critique à laquelle l'utilisateur destinataire ne doit pas avoir accès. Une fois que le demandeur a envoyé sa demande de vote à la carte (1), celle-ci chiffre donc la requête (d'identifiant *Excl*) avec l'identité de l'utilisateur visé (2) et l'envoi(3)(4)(5). La carte réceptrice traite la requête en incrémentant le compteur de vote d'exclusion reçu (6). L'identité de l'émetteur est enregistrée pour qu'il ne puisse pas en envoyer plusieurs. Si le seuil maximum est atteint, la carte marque l'utilisateur comme exclu (champ de l'OPG) et lui renvoie un message lui indiquant son éviction (4).

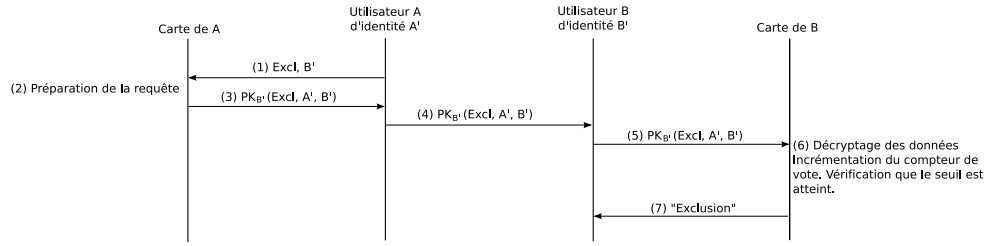


Fig. 40 – Echange d'un message de groupe dans l'application SManet

e) Participation au groupe.

Il s'agit maintenant de la présentation de la preuve d'admission lors d'une communication entre membres du groupe. Rappelons qu'un message peut avoir comme destinataire un ou plusieurs individus, ou encore être adressé à l'ensemble des membres du groupe.

Problématiques et existant. Nous traitons ici des types de communication individuel et de groupe.

Communications individuelles. Pour ce type de communication, il faut fournir aux nœuds une clé individuelle authentifiée (si elle ne leur a pas été remise à l'admission). Une telle clé permet de communiquer en assurant la confidentialité, l'authentification, l'intégrité et la non-répudiation entre deux nœuds ou plus. Dans les réseaux classiques on met pour cela en place une PKI (cf. 1.4.3(b) page 41). Dans les MANets, les solutions décentralisées ou auto-certifiées sont applicables mais toutes présentent des contraintes :

- la méthode de décentralisation de l'autorité de confiance par les « threshold secret sharing » est présentée en 1.4.1(d). De nombreux travaux utilisent une décentralisation partielle [162, 167], d'autres une décentralisation totale [97, 103]. Ces solutions nécessitent tout de même une phase importante d'initialisation pour répartir la clé publique globale sur plusieurs nœuds.

- les approches sans autorité de confiance [37, 66, 83] sont basées sur les solutions de type PGP. Nous avons vu en 1.4.3(b) (page 42) que ces méthodes sont « peu sûres » car elles reposent sur l'établissement de taux de confiance eux-mêmes non sûrs. Elles sont en fait destinées à certains contextes.

– les solutions de pré-distribution de clés ont surtout été adaptées aux MANets composés de capteurs [41, 52, 53, 59]. Ces méthodes requièrent une importante préparation des pools de clés par une organisation humaine (cf. 1.4.3(b), page 42).

Une solution pour des MANets très particuliers formant un réseau domestique est présentée dans [143]. Le principe est qu'un dispositif reconnaît comme « mère » le premier autre dispositif qui lui envoie une clé secrète. Toute la sécurité est basée sur le contexte « domestique » dans lequel tous les nœuds sont considérés comme honnêtes. L'échange de la première clé entre chaque paire est considéré comme sûr car réalisé physiquement par l'utilisateur ou encore par un lien infrarouge lorsque les dispositifs sont suffisamment proches. Cette solution n'est valable que dans ce contexte précis.
Dans [79, 126], où des alternatives existantes sont décrites, il est aussi proposé une solution permettant d'étendre la confiance établie entre deux dispositifs à l'ensemble d'une communauté dont les membres sont eux-mêmes liés par des relations de confiance.

Communications de groupes. Les communications de groupes nécessitent la distribution d'une clé de communication dont tous les membres du groupe et seuls eux doivent disposer.

Dans la section 1.4.3(c) (page 43), nous avons évoqué trois types de solutions pour une telle distribution (centralisée, décentralisée et non centralisée). Or les deux premiers types de solutions sont basés sur des contrôleurs de groupes qui sont des points centraux ayant une tâche particulière et plus lourde que celle des autres nœuds. Ces solutions ne sont pas adaptées aux MANets car elles ne supportent pas une trop grande « dynamicité » de la topologie et ne gèrent pas les partitionnements (le contrôleur de groupe principal est le seul à avoir certaines informations).

Dans le dernier type de solution, la clé de groupe est générée par les membres de ce groupe eux-mêmes, ce qui libère du besoin d'un point central administré. Ces solutions nécessitent tout de même la spécialisation de l'un des membres pour la distribution de la clé finale, ce qui est par ailleurs une tâche lourde. De plus, la plupart des solutions organisent un arrangement précis des nœuds (en série, en hypercube, etc) représentant une phase d'initialisation importante et nécessitant une faible mobilité à ce moment là.

Les études spécialement réalisées pour les communications de groupes dans les réseaux MANets se basent sur l'organisation en clusters. Par exemple, dans l'approche des *pebblenets* [16], le réseau est régulièrement séparé en *clusters* dans lesquels est élu un *leader*. Un de ces *leaders* est à son tour élu pour dériver la clé secrète principale et l'envoyer aux autres *leaders*, eux-mêmes chargés de la distribuer aux nœuds de leur *cluster*. Dans [58], le principe des *clusters* vu avec *pebblenets* est réutilisé avec une autorité de certification distribuée.

La création et la distribution des éléments nécessaires à la participation au groupe doit être faite par le module. La certification de ces informations repose sur le fait que les échanges se déroulent via un canal accessible uniquement par les modules. Il s'agit en fait du besoin 2.4 (page 73).

Protocole. Echange de données point à point (fig. 41) Un utilisateur peut vouloir communiquer avec l'un des éléments de son groupe en particulier. Les échanges peuvent être de simples messages à lire par l'individu cible ou encore un élément secret telle qu'une clé de session. Pour ce dernier cas, nous mettons en place un moyen d'échanger une clé de session et de la stocker sur la carte à puce de sorte qu'elle soit non répudiable (aucun des nœuds ne peut l'effacer ou la modifier).

La clé de session est une donnée critique, il faut donc qu'elle soit chiffrée avec la clé publique qui compose l'identifiant de la cible (cf. 2.2.4 page 58).

Tout d'abord, l'utilisateur envoie une requête à sa carte avec l'identifiant de l'interlocuteur et du groupe concernés (1). La carte fait les vérifications nécessaires (existance des identifiants, connexion au groupe, etc.)(2) et garde en mémoire l'association de la clé de session avec les deux identités concernées jusqu'à recevoir la réponse de l'interlocuteur (3). Au message de base (ayant pour identifiant de requête *RExch*), la carte ajoute la clé publique du groupe, sa propre identité publique (clé publique et identifiant mais qui, pour simplifier, est représenté ici par $PK_{A'}$) et la clé de session (4). Ce message est envoyé, puis reçu par la carte cible (5)(6)(7). Celle-ci déchiffre le message et vérifie qu'il lui est bien adressé (8). Elle transmet ensuite au niveau utilisateur l'identifiant reçu (9). L'utilisateur décide s'il accepte la communication ou non et remet sa réponse à la carte (10). Si la réponse est positive, la carte stocke définitivement l'association de la clé de session avec les identités concernées (11). Le choix doit être fait par l'utilisateur et non pas automatiquement pour ne pas que la carte accepte toutes les requêtes et sature sa mémoire. Elle prépare ensuite la réponse en ajoutant au message de base la clé publique du groupe et un indicateur de son accord. Elle chiffre ensuite ce message (qui a pour identifiant de requête *RespExch*) avec la clé publique associée à l'identité de A (12). La carte initiatrice reçoit le message (13)(14)(15) et le décrypte (16). A son tour, cette carte stocke définitivement l'association de la clé de session avec les identités concernées (17) et renvoie la réponse à l'utilisateur (18).

Si le premier échange entre les utilisateurs échoue (6), il n'y a aucune conséquence et la requête devra être relancée. La carte garde en mémoire temporairement la clé de session tant que la réponse ou un *time out* (qui se base sur le nombre de messages échangés) n'est pas reçu.

Il se passe exactement la même chose si le second échange (14) échoue. Dans ce cas là, la carte de B' aura enregistré l'association mais parce que son utilisateur l'aura accepté. Si A (qui n'a pas enregistré la réponse et ne sait donc pas que B, lui, l'a enregistrée) relance la requête, la carte de B répondra positivement directement à A sans le demander de nouveau à l'utilisateur. Si la clé de session n'est plus la même, par contre, la question sera reposée à B.

Remarque : dans le cas où le message n'est qu'un texte à lire par l'entité ciblée, le protocole est le même sauf qu'il n'y aura pas d'enregistrement du message et le récepteur ne retournera aucune réponse.

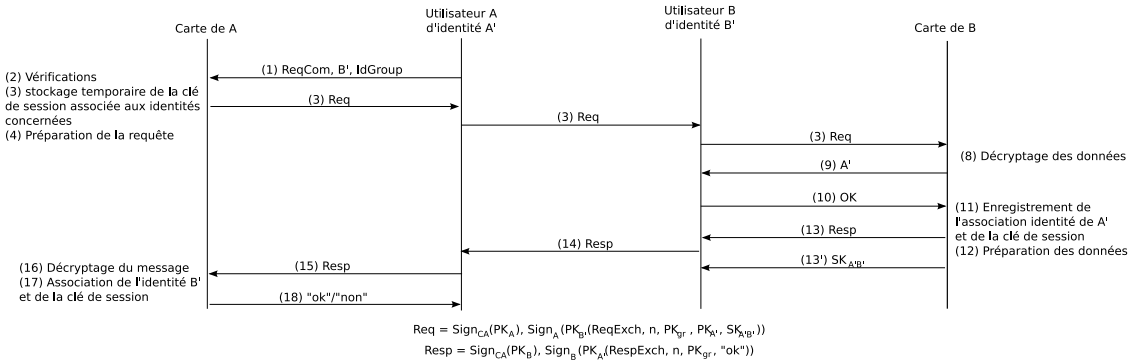


FIG. 41 – Echange d'une clé de session dans l'application SManet

Communications de groupe (fig. 42) Pour les communications de groupes, il suffit de transmettre à sa carte la requête appropriée avec l'identité du groupe choisi et le message à envoyer avec pour identifiant de requête *CommG* (1). Elle le crypte avec la clé du groupe (2). Les cartes reçoivent le paquet (2)(3)(4) et après déchiffrement (5), seules celles qui font partie du groupe et y sont actives le traitent (6). Le message est finalement transmis au niveau utilisateur (7).

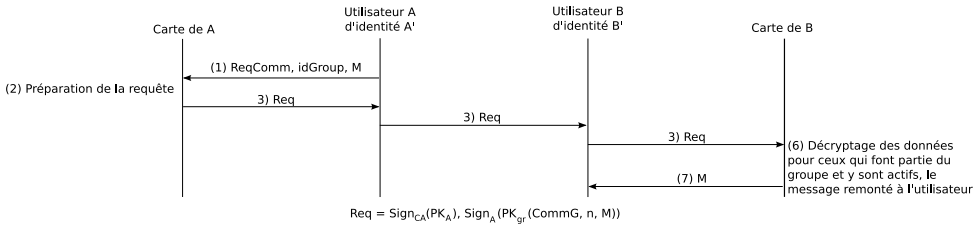


FIG. 42 – Echange d'un message de groupe dans l'application SManet

Schéma de la solution proposée pour la gestion des groupes

La gestion sécurisée des groupes est entièrement effectuée en local sur chaque nœud comme cela est décrit sur la figure 43

Conclusion

Les solutions existantes pour mettre en œuvre la sécurité des communications dans les MANets sont basées soit sur une centralisation, soit sur une planification. Ceci ne convient pas pour des MANets purs totalement spontanés.

La carte à puce nous permet d'« embarquer » un module administrateur chargé de tous les contrôles nécessaires pour assurer la sécurité des communications. Ce module apporte une forte sécurité à la gestion des identités et à la formation de groupes en contrôlant les opérations sensibles (distribution d'éléments secrets), mais aussi des problématiques spécifiques à notre contexte (expression du but, gestion des moyens d'autorisation et d'admission, gestion des conditions d'entrée).

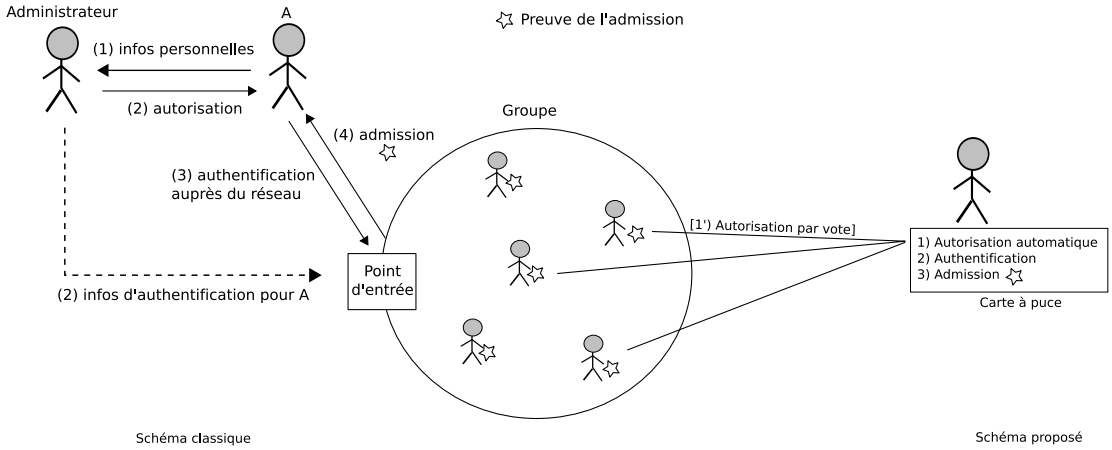


Fig. 43 – Comparaison de la solution proposée pour la gestion des groupes avec les solutions classiques

Toutes ces opérations se font sur le nœud lui-même mais de manière indépendante de l'utilisateur et protégée de celui-ci. L'utilisateur reste libre des contraintes habituelles, le système ne requérant lui-même aucune préparation une fois les cartes sorties de l'usine.

En résumé, on peut dire que la carte à puce est donc un administrateur idéal, contrôleur des données et des protocoles. Grâce à elle, aucune centralisation sur aucun nœud n'est nécessaire, aucune tâche n'est confiée à un serveur particulier, que ce soit avant la création ou pendant la vie du réseau. Chaque carte est garante de la politique de gestion des identités, de l'expression des caractéristiques des groupes et de la gestion des accès à ces groupes.

Ainsi, nous permettons à un réseau de type MANet d'exploiter tout son potentiel en offrant la possibilité aux nœuds et/ou à leurs détenteurs d'exprimer leur but, leurs possibilités, leurs caractéristiques. Nous avons aussi permis la distribution de ces informations aux autres afin de générer des communications et donc un travail collaboratif indépendemment de toute organisation humaine pré-établie.

Chapitre 3

Une option de sécurité supplémentaire : élection d'un leader

Sommaire

Dans le chapitre précédent nous avons détaillé la solution que nous proposons et qui est basée sur un module administrateur. Cette solution permet en particulier d'assurer la *forward secrecy* c'est-à-dire l'impossibilité pour un membre qui a quitté le groupe d'accéder aux données échangées après son départ. En revanche, notre solution en l'état actuel ne permet pas d'assurer la *backward secrecy* i.e. l'impossibilité pour un membre d'accéder aux communications antérieures à son entrée dans le groupe. Ainsi, si un utilisateur enregistre des communications d'un groupe dont il n'est pas membre, et si ensuite il est accepté dans ce même groupe, il pourra avoir accès aux échanges enregistrés.

Dans ce chapitre, nous proposons un module supplémentaire permettant de mettre en place la *backward secrecy*. Nous avons pour cela développé un algorithme d'élection d'un *leader* qui, une fois élu, aura pour rôle de dater les messages. Nous donnons ensuite la possibilité au module administrateur de refuser à son utilisateur la lecture de messages

antérieurs à sa date d'entrée dans le groupe.

Cette sécurité supplémentaire sera optionnelle dans l'application que nous proposons. Outre l'aspect sécuritaire de ce système, la mise en place d'une telle option nous sert à tester l'utilisation et l'efficacité des modules de sécurité précédemment établis (fig.44). Le développement d'un algorithme d'élection de *leader* nous permet aussi d'approndir notre étude sur le développement d'algorithmes distribués respectant les caractéristiques des MANets purs : prise en compte de contraintes telles que l'égalité des utilisateurs (la même charge de travail doit être demandée à tous les utilisateurs), gestion optimisée du nombre de messages, gestion optimisée des calculs pour économiser les ressources, développement basé sur une connaissance locale uniquement, etc.

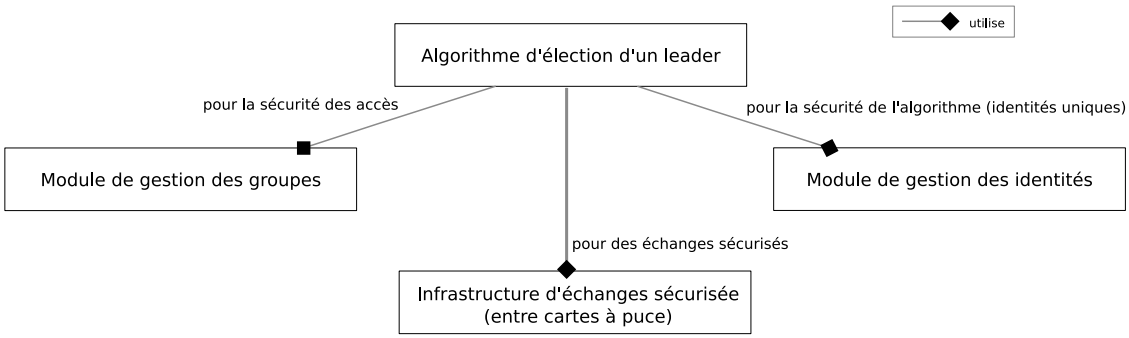


Fig. 44 – Relations entre l'algorithme d'élection du *leader* - module administrateur

Nous commençons ce chapitre par une présentation générale du domaine de l'élection d'un *leader*. Ensuite, nous exposons l'utilisation et le fonctionnement de l'algorithme que nous proposons. Enfin, nous mettons en place une modélisation de cet algorithme afin d'en établir la validité.

3.1 Présentation du domaine

Dans cette section nous présentons la notion de *leader* et d'élection d'un *leader*.

3.1.1 Introduction à la notion de *leader*

a) Définition et utilisation

Le *leader* d'un groupe est un nœud qui va avoir une mission particulière au sein de ce groupe. Nous considérons dans ce travail deux types d'utilisation d'un *leader*. Il peut être :

1. un point de centralisation d'informations : le *leader* centralise les données des autres utilisateurs pour les soumettre à un traitement. Il faut alors respecter quelques conditions pour rester cohérent avec l'environnement des MANets purs dans lequel nous travaillons :
   – les ressources des dispositifs utilisateurs étant plutôt faibles, les données reçues par le *leader* doivent être limitées en taille et ne peuvent être stockées que temporairement ;

- la tâche du *leader* ne doit pas être trop lourde pour ne pas trop ralentir les autres tâches du dispositif ni utiliser trop de batterie ;
- il ne faut pas que l'inégalité entre les nœuds, du point de vue de la charge de travail, soit trop importante : aucun utilisateur ne doit dépenser beaucoup plus d'énergie et de ressources que les autres.

Malgré ces précautions, ce mode d'utilisation impose au point central sa disponibilité (ce qui est une contrainte) et, nécessairement, une tâche plus importante.

2. un point de repère : dans ce cas, le *leader* ne centralise aucune donnée. Il a uniquement pour rôle d'envoyer des informations de référence devant être connues de tous (par exemple une « date » dans notre cas). La seule contrainte est alors que la gestion de ces informations et la génération des requêtes nécessaires ne soient pas trop lourdes.

b) Exemples d'utilisation de *leaders*

Les *leaders* sont le plus souvent utilisés par des applications de sécurité et/ou d'organisation. Pour les réseaux ad hoc, les domaines principaux dans lesquels les *leader* sont employés sont les suivants :

- organisation des réseaux en *clusters* [16, 19, 164] : il s'agit de constituer des sous-groupes de nœuds proches les uns des autres, chaque sous-groupe étant représenté par un *leader*. Les *leaders* de *clusters* ou *clusterhead* peuvent avoir plusieurs rôles : allocation d'adresse pour les nœuds du *cluster*, simplification du routage [19], organisation des communications, réplication de données [164] ;

- gestion des groupes [17], le *leader* joue le rôle de contrôleur des accès au groupe (génération et maintien des clés) dont il est chargé ;

- maintient d'un arbre couvrant. Un arbre couvrant peut être utilisé par exemple pour diminuer les redondances lors de diffusion de messages (les messages ne sont réémis que par les nœuds de l'arbre), etc..

- adressage des nœuds. Dans [65] le *leader* identifie les sous-réseaux connexes, fournit des adresses aux nœuds et en maintient une liste complète.

3.1.2 L'élection d'un *leader*

a) Définition et principes

L'élection d'un *leader* est le procédé par lequel un des membres du groupe considéré va être choisi. Une élection est basée sur la comparaison d'un ou de plusieurs critères que chaque nœud possède mais dont la valeur peut être différente pour chacun. Il existe deux types de méthodes de comparaison :

1. la méthode *extrema finding* : le nœud ayant la plus grande ou la plus petite valeur de critère est élu. Le critère peut alors être en relation avec les nœuds (identifiant, capacité de calcul, consommation d'énergie [60, 153]), l'environnement (nombre de

liaisons avec des nœuds) ;

2. la méthode *non extrema finding* : entre deux nœuds, on en choisit un au hasard. Dans cette méthode il n'est plus possible d'adapter l'élection en fonction des dispositifs et de leur environnement.

b) Algorithmes existant pour l'élections d'un *leader* pour les MANets

Le problème de l'élection d'un *leader* dans un système distribué a été très étudié dans le cadre des réseaux statiques et dynamiques. Cependant, les algorithmes résultants sont peu adaptés au contexte des MANets purs (on trouvera les détails en [153]). Quelques propositions ont été cependant élaborées spécialement pour les MANets purs, dont les principaux sont les suivants :

- l'algorithme proposé dans [105] est basé sur TORA, algorithme de routage permettant de trouver toutes les routes possibles entre un nœud et sa cible. Le principe est le suivant : l'algorithme attribue un sens aux arêtes de manière à ce qu'elles soient orientées vers le *leader*. Lorsque deux nœuds se connectent l'un à l'autre, le nœud de *leader* le plus grand adopte le *leader* le plus petit et l'arête qui relie ces deux nœud est orientée vers le nœud de *leader* gagnant (c'est lui qui offre le chemin).
  Par contre, lorsqu'un nœud détecte une rupture de son unique chemin vers le *leader* (il n'a plus aucune arête sortante), il lance une recherche parmi ses voisins qui eux-mêmes propagent la recherche. Finalement, si aucun nœud n'a de chemin vers le *leader*, le nœud ayant détecté la rupture est élu (comme *leader*). L'algorithme n'est donc pas *extrema finding* puisqu'on ne se base plus sur la valeur du critère des nœuds. Le maintien de toutes les routes vers le *leader* constitue une surcharge car il entraîne un grand nombre d'échanges. Cette méthode d'élection est plus particulièrement adaptée si par la suite on souhaite réaliser un routage (sinon il n'est pas forcément nécessaire de connaître toutes les routes possibles vers le *leader*). De plus, la validité de cet algorithme n'est pas prouvée et il n'apparaît pas comme évident que tous les cas soient traités ;

- dans [151] deux algorithmes prenant en compte la sécurité sont présentés. On suppose que les nœuds possèdent des clés publiques pour le chiffrement sécurisé des échanges. Le principe de l'algorithme est différent du précédent : chaque élection repose sur la construction progressive d'un arbre couvrant. Les nœuds prennent pour parent le premier nœud dont ils reçoivent un message d'élection. S'ils ont des voisins encore « orphelins » ils attendent, sinon se sont des feuilles et ils renvoient à leur parent leur identité et leur valeur de critère d'élection. Les parents réalisent une élection partielle en utilisant les messages reçus et leur propre valeur. Le nœud ayant initialisé l'élection récupère les messages de tous ses fils et procède à l'élection finale. Ensuite, il diffuse l'identité du nouveau *leader*.
  La différence entre les deux algorithmes est la fonction d'évaluation de la valeur du critère des nœuds : dans le premier, cette fonction retourne toujours la même valeur pour un même critère, dans le deuxième, des paramètres environnementaux entrent en compte et un même critère peut être évalué différemment sur des nœuds différents.

Ces deux algorithmes nécessitent un système distribué synchrone et un seul changement topologique à la fois est géré (les nœuds doivent rester statiques le temps de l'élection). De plus, pour que l'on puisse repérer la disparition du *leader*, celui-ci envoie des *heartbeat* régulièrement. Plusieurs nœuds peuvent se rendre compte que le *leader* a disparu et ainsi plusieurs procédures d'élection peuvent être lancées. Une méthode gère ce problème après coup mais cela génère tout de même un important nombre de messages inutiles ;

- dans [152, 153], un article des mêmes auteurs, un algorithme reprenant le principe des deux précédents est proposé. Il ne nécessite plus un environnement synchrone et traite plusieurs changements topologiques à la fois. Par exemple, pour repérer qu'un nœud sort de son îlot pendant une élection, des messages de découverte des voisins sont envoyées régulièrement. Pour la fusion de deux îlots, un des deux *leader* est élu et l'information est diffusée à l'ensemble de l'îlot de *leader* perdant. Dans le cas où deux îlots fusionnent alors qu'une procédure d'élection est en cours dans au moins l'un des deux îlots, il faut attendre que toute élection soit finie avant de procéder comme pour une fusion simple.
  Les problématiques dues au système de *heartbeat* sont les mêmes que celles des algorithmes présentés dans [151] ;

- dans [104] l'algorithme présenté dans [153] est modifié afin de réduire le nombre d'échanges : au lieu d'élire un unique *leader* l'algorithme construit une liste de cinq nœuds dont le premier est *leader* et les autres sont ses remplaçants, dans l'ordre, au cas où il disparaîtrait. Si une partition a lieu, les nœuds de la liste ne faisant plus partie de l'îlot sont mis en fin de liste (une refusion étant possible, voire problable). C'est une amélioration intéressante qui permet de ne pas relancer toute la procédure chaque fois qu'un nœud perd le contact avec son *leader*. Cependant, le système de *heartbeat* existe toujours et pose les mêmes problèmes que dans [153].
- dans [78] l'algorithme présenté est *extrema finding*. Son approche est très différente des deux précédentes. Les nœuds se déplacent dans des espaces organisés en polyhèdres adjacents (dont ils connaissent les dimensions) et on suppose qu'ils n'appartenniennent qu'à un polyhèdre à la fois. Les nœuds doivent aussi connaître la distance qu'ils parcourent et compter le nombre d'autres nœuds qu'ils « croisent ». Lorsqu'un nœud change de polyhèdre et que dans ce nouveau polyhèdre il croise un autre nœud, ceux-ci échangent leurs identités. Celui qui à l'identité la plus grande reste candidat et reçoit le compteur de l'autre nœud qu'il ajoute au sien. Le nœud perdant devient inactif et ne participe plus à l'algorithme.

3.2 Utilisation et fonctionnement de l'algorithme d'élection de leader proposé

3.2.1 Rôle du *leader* : datation des messages

Discussion sur le rôle « central » du *leader*. Dans notre solution, le *leader* est un point de repère (cf. 3.1.1(a) page 84) et non pas un point de centralisation : il est

chargé d'incrémenter un compteur représentant une « date » de référence du réseau. Les autres nœuds utilisateurs n'ont aucun devoir envers lui, n'ont pas besoin de le consulter et reçoivent simplement des informations de sa part. De plus, nous verrons par la suite que la tâche que nous attribuons à ce *leader* n'est pas plus lourde que celle réalisée par les nœuds non *leader*.
Ce type d'utilisation du *leader* ne constitue donc en rien une contrainte et il n'est pas en désaccord avec les caractéristiques des MANets purs non planifiés.

Principe général. Le principe est simple : les messages échangés entre des membres d'un groupe sont datés et chaque nœud connaît sa date d'entrée dans le groupe. Un utilisateur ne peut avoir accès qu'aux messages marqués d'une date postérieure ou correspondant à sa date d'entrée. Pour que ceci fonctionne, la date ne suffit pas, il faut aussi ajouter de quel îlot les messages proviennent. En effet, si un nœud stocke des messages alors qu'il n'est pas inscrit au groupe et qu'ensuite il entre dans un îlot dont la date est antérieure à celle des messages enregistrés, il pourra y accéder.

Principe de diffusion de la date. La date est un paramètre de l'OPG du groupe dans lequel on se trouve. Avant d'entrer dans ce groupe, cette date a une valeur indéfinie. Il n'est alors pas possible pour le nœud d'accéder aux messages. A la première réception d'un message daté, la carte à puce met à jour le champs « date » de l'OPG du groupe concerné. Ensuite, à chaque réception d'un message de date postérieure à celle de l'OPG, la carte remet à jour ce champs.

Optimisation des échanges. La propagation de la date (qui est un simple compteur) à tous les nœuds d'un îlot ne doit pas générer un trafic trop important. Pour cela la diffusion se fait par l'intermédiaire des requêtes/réponses envoyées régulièrement dans le cadre de la procédure de découverte des voisins. Ainsi, il n'y a aucun surcoût en terme de nombre de messages. De plus, la tâche de mise à jour est la même pour tous les nœuds y compris pour le *leader*, la seule différence étant que c'est le *leader* qui est chargé de faire avancer la date donc il incrémente,de sa propre initiative, le compteur qui « implémente » le temps. En pratique, cette incrémentation a lieu tous les n messages envoyés par le *thread* de découverte des voisins.
La distribution du nom de l'îlot (nom du *leader*) se fait exactement de la même manière que pour la date, en utilisant les échanges nécessaires à la découverte des voisins.

Remarque : sur les cartes à puce que nous utilisons, le compteur est de type *short* et offre donc 65536 valeurs possibles (de -32768 à +32767 et la valeur 0). Le compteur est ainsi incrémenté à chaque renouvellement de date. Si le compteur atteint sa valeur maximale (+32767, il est réinitialisé et reprend à sa première valeur (-32768). Ce compteur est limité mais nous considérons le nombre de valeurs possibles suffisant pour des réseaux MANets constitués pour des durées relativement courtes. C'est un point important à prendre en compte car si l'on revenait sur une même date (par un modulo une fois la valeur maximale atteinte) il serait possible de rejouer d'anciens messages. Il sera toutefois possible d'utiliser un compteur beaucoup plus grand sur des cartes à puce ayant des ressources relativement importantes et lors de tests à grande échelle.

Cas d'un partitionnement et d'une fusion. Si le réseau est partitionné et que les dates ne sont plus les mêmes dans les différents sous-réseaux, lors d'une fusion, c'est la date la plus récente qui est adoptée : si la plus ancienne était adoptée on aurait le risque d'avoir des nœuds avec une date d'entrée postérieure à la date courante et qui seraient donc dans l'incapacité de lire les messages courant alors qu'ils font partie du groupe. Un message reçu d'un autre îlot et de date antérieure à celle de l'OPG n'est pas traité.

3.2.2 Principe de l'algorithme d'élection de leader proposé

Principe général

L'algorithme d'élection d'un *leader* que nous avons développé permet d'avoir un unique *leader* par îlot de nœuds d'un même groupe. Le principe de cet algorithme est inspiré de celui présenté dans [105] ; il repose sur la connaissance du plus court chemin jusqu'au *leader*.

Remarque : cet algorithme n'est pas *extrema finding* car lors de la détection d'un îlot, c'est le nœud qui a détecté la rupture qui devient le *leader*. Dans notre cas précis, nous considérons que tous les nœuds doivent avoir la même charge de travail donc il n'est pas utile de chercher à élire un nœud en fonction de ses ressources. Il ne s'agit pas non plus de centraliser des informations d'autres nœuds mais de diffuser une information donc il n'est pas non plus nécessaire de considérer le nombre de liens avec d'autres nœuds dans les critères d'élection.

L'ensemble des plus courts chemins de tous les nœuds vers le *leader* forme un arbre couvrant (fig. 45). Pour constituer ce plus court chemin, chaque nœud a connaissance de deux éléments principaux :

- le nombre de sauts jusqu'au *leader* (chaque nœud intermédiaire comptant pour un saut). Le *leader* a pour longueur de chemin 0, et les fils du *leader* ont pour longueur de chemin 1 ;

- son prédécesseur direct, c'est-à-dire le nœud avant lui dans le chemin, i.e. celui qui lui offre le plus court chemin jusqu'au *leader*. Comme nous le verrons, tout nœud i adoptera comme prédécesseur le nœud voisin v_i ayant le nombre de sauts $n(v_i)$ le plus petit et mettra son propre chemin à $n(v_i) + 1$.

Le plus court chemin est mis à jour sur les nœuds concernés à chaque évènement topologique tel qu'une connexion, une déconnexion ou un changement concernant le chemin vers le *leader* d'un voisin.

L'algorithme comporte plusieurs phases pouvant se dérouler simultanément dans des parties différentes ou dans une même zone d'un réseau. Ces phases sont les suivantes (elles sont détaillées à l'aide des règles de réécriture en 3.3.3(b) page 96) :

1. phase de mise à jour suite à l'apparition d'un nœud dans un voisinage : si le nouveau nœud offre un chemin plus court vers le *leader* à un de ces voisins, ce voisin se met à jour. A l'inverse, si l'un des voisins offre un chemin plus intéressant au nouveau nœud, ce dernier l'adopte ;

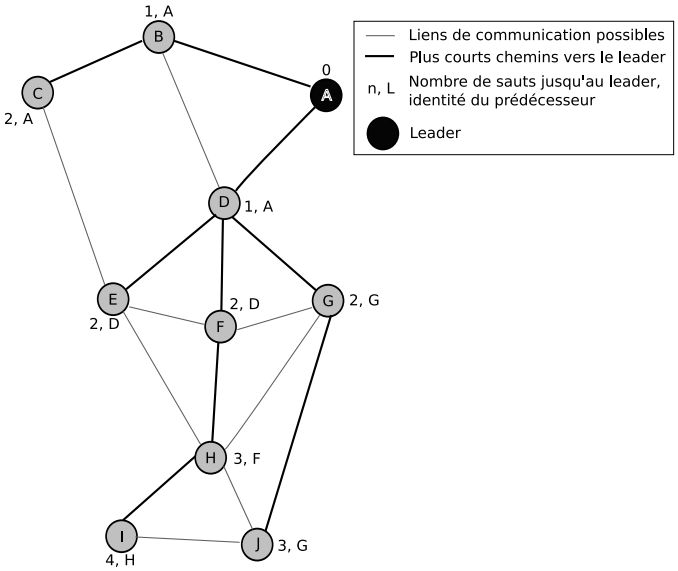


FIG. 45 – Exemple de fonctionnement de l'algorithme d'élection d'un *leader* proposé

2. phase de mise à jour suite à la fusion de deux îlots : tous les nœuds de *leader* ayant une valeur de critère plus grand adoptent le *leader* de valeur de critère plus petit et mettent à jour leur chemin (nombre de saut et prédécesseur) en conséquence ;

3. phase de mise à jour suite à une perte de chemin vers le *leader* : les nœuds doivent vérifier si un autre chemin vers le *leader* existe. Si leurs voisins directs ne leur offrent pas de nouveau chemin, ils entrent dans une phase de recherche ;

4. phase de recherche d'un nouveau chemin : les nœuds interrogent leurs fils à propos d'un éventuel chemin vers le *leader* et se mettent dans un état indéfini (cf. 3.3.3(a) page 95) en attendant une réponse. Si aucun fils ni voisins ne leur offre de chemin, les nœuds se mettent à l'état dit *nul* (cf. 3.3.3(a) page 95) ;

5. phase de stabilisation : elle correspond au moment où il n'y a ni connexion ni déconnexion. L'îlot a un unique *leader* et tous les nœud connaissent le plus court chemin jusqu'à celui-ci.

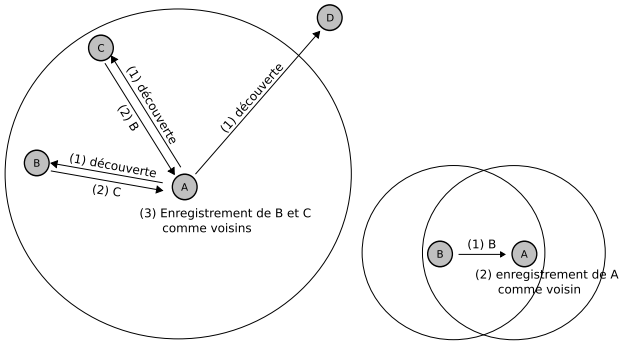
Un nœud peut être le voisin d'un autre (ils sont chacun dans la portée d'émission l'un de l'autre) et pourtant ne pas le détecter lors d'une découverte des voisins à cause des problèmes de fiabilité des ondes radio (cf. 1.2.4(a) page 16) ou suite à une perte de paquets. Cependant les conséquences ne sont pas graves pour l'algorithme car dans tous les cas chaque îlot aura un unique *leader*. Si un voisin offrant un plus court chemin n'est pas détecté lors d'une découverte, il le sera lors de la suivante ; dans le cas contraire cela signifie que le nœud voisin est très difficilement accessible et donc il est préférable qu'il ne soit pas considéré comme un prédécesseur potentiel.

De la même manière, la perte d'un paquet peut engendrer l'illusion de la disparition d'un *leader*. Cependant, à la prochaine découverte du voisinage l'erreur sera corrigée ; dans le cas contraire cela signifie que le lien considéré n'est de toutes façons pas suffisamment fiable pour être pris en compte.

A propos des liens de communication

Il existe deux modes de découverte des voisins :
- réactive (fig. 46(b)) : chaque nœud envoie régulièrement des messages pour indiquer sa présence. Les autres nœuds reçoivent cette information et la traitent (réaction à la réception d'un message).
- proactive (fig. 46(a)) : les nœuds lancent régulièrement une requête à laquelle tous les nœuds la recevant répondent par un message indiquant leur présence.

La technologie sans fil que nous employons offre des portées d'émission variées. Ainsi les liens de communication entre deux nœuds peuvent être de deux types :
- symétriques : si un nœud i est à portée d'un nœud j, alors j est à portée de i. Dans ce cas, il est possible de faire une découverte des voisins réactive et proactive.
- asymétriques : un nœud i peut être à portée d'un nœud j sans que l'inverse soit forcément vrai. Il n'est alors pas possible d'avoir une approche réactive pour la découverte des voisins : en effet, si un nœud j envoie un message pour se signaler, un nœud i le recevant ne peut pas être sûr que lui peut l'atteindre et donc qu'il s'agit bien d'un voisin (au sens symétrique du terme).



(a) Découverte proactive (b) Découverte réactive

Fig. 46 – Méthode de découverte des voisins

Remarque : dans notre contexte, les voisins d'un nœud sont définis comme ceux avec qui il est possible d'établir une communication symétrique.

Le plus souvent, dans les solutions développées, les portées des communications sont considérées comme symétriques. Or nous sommes dans un cadre pratique où les liens de communication sont asymétriques et nous les considérons donc comme tels. Comme nous l'avons dit, une approche réactive (découverte à la réception d'un message) ne nous permet pas d'être sûr qu'un nœud est un voisin. Une approche uniquement proactive est trop

lente : pour prendre connaissance de l'apparition d'un nœud il faut attendre le lancement de l'opération de découverte suivante. Pour améliorer cela, nous adoptons un mélange de découverte réactive et proactive : un nœud i procède à une découverte de son voisinage de manière proactive, c'est-à-dire en envoyant régulièrement des requêtes et en collectant les réponses. En plus de cela, lorsqu'il reçoit la requête d'un nœud j qui n'est pas dans son voisinage (lien de communication asymétrique), il avance la procédure de découverte pour savoir si ce nœud est un voisin ou non. Si j n'est pas un voisin, à chaque requête de celui-ci i va lancer une découverte des messages. Si plusieurs nœuds sont dans le cas de j, i risque de réaliser un nombre excessif de découvertes du voisinage. Pour éviter cela, un nœud dont on reçoit les messages mais qui n'est pas un voisin est marqué de manière à ce qu'aucune découverte ne soit lancée à nouveau (sauf s'il disparaît du voisinage un moment et qu'il y revient). Ce système nous permet une mise à jour de l'arbre couvrant plus rapide.

3.2.3 Sécurité de l'algorithme

Dans notre solution, l'algorithme s'exécute sur des cartes à puce et les échanges se font par des canaux sécurisés. Aucune donnée n'est donc modifiable illégalement. Les identités sont sûres (grâce au module de gestion des identités) et les communications au sein du groupe aussi (grâce au module de gestion des groupes).

Un problème de sécurité pourrait être l'interception de messages avant qu'ils ne soient remis à la carte à puce. Cependant, le chiffrement des communications permet d'empêcher une telle attaque car il n'est plus possible de reconnaître et donc de filtrer uniquement ces messages. En revanche on pourrait éventuellement les reconnaître à leur taille, il faudrait donc s'assurer que tous aient la même par l'incorporation de bits aléatoires par exemple.

Les problèmes de sécurité ne peuvent donc intervenir qu'à partir du moment où des données proviennent de ou partent vers l'extérieur du réseau de cartes à puce (voir exemple pour la problématique de la découverte des voisins).

Sécurité du critère d'élection. Si le critère d'élection est un élément provenant de l'extérieur de la carte à puce, il est sensible à des attaques de l'hôte lui-même ou d'un nœud malveillant. La modification de la valeur du critère peut constituer une attaque ayant pour but d'augmenter ses chances d'être élu (pour avoir un plus grand contrôle sur le groupe) ou inversement de les diminuer (pour économiser de l'énergie si le *leader* a une lourde tâche).
Dans notre solution, le critère d'élection est l'identité qui est générée sur la carte. Il n'y a donc pas de problème de sécurité à ce niveau là. Pourtant il serait intéressant pour un créateur de groupe de pouvoir choisir le critère d'élection qu'il souhaite. Pour cela il faudrait pouvoir installer du code sur la carte à puce depuis l'intérieur même de la carte. Ce code (chargé de récupérer la valeur du critère sur chaque nœud) serait envoyé avec l'OPG du groupe.

Problématique pour la découverte des voisins. Les OS de cartes à puce que nous avons employés ne proposant pas de *threads*, la découverte régulière des états des voisins

doit utiliser un *thread* de niveau utilisateur. Des attaques sont alors possibles : un utilisateur pourrait bloquer les requêtes des *threads* (pour économiser de l'énergie par exemple). Une conséquence pour l'algorithme serait que l'arbre des plus courts chemins soit faussé : le nœud qui ne lance pas de requête pour découvrir son voisinage pourrait ne pas connaître suffisamment rapidement l'état de ses voisins (leur plus court chemin), avoir lui-même un mauvais plus court chemin et ainsi, donner de mauvais renseignements à ses propres voisins.

Pour pallier ce problème, nous avons mis en place un système permettant à la carte de savoir si l'utilisateur lui envoie bien toutes les informations nécessaires en temps voulu : la carte marque les réceptions de requête de voisins et vérifie, que pour un nombre n choisi de requêtes reçues d'un même voisin, elle a bien envoyé une requête. Si elle se rend compte qu'aucune requête de découverte du voisinage ne lui a été demandée, elle bloque immédiatement l'accès à ce groupe pour son hôte. Ce bloquage doit pouvoir être levé au cas où le non-envoi des éléments serait dû à un disfonctionnement non volontaire (par exemple, une attaque des requêtes de recherche pour provoquer le bloquage de la carte).

Faiblesse de l'algorithme Notre solution pourrait présenter une faiblesse de sécurité liée à l'asynchronisme de la diffusion de l'information de datation sur tout un îlot. En effet, à un moment donné, dans un îlot, la partie haute de l'arbre (proche du *leader*) peut être à un instant t' alors que la partie basse peut être à un instant t $<$ t' (l'information de datation n'est pas encore parvenue jusque là). Imaginons le scénario suivant : un utilisateur I non membre enregistre[1] les messages au moment où le *leader* est à la date t. Le *leader* passe ensuite à la date t' mais la propagation jusqu'aux feuilles de l'arbre prenant un certain temps le nœud I peut devenir membre (en bas de l'arbre) en apparence à la date t et ainsi avoir accès aux messages qu'il a enregistrés.
Cependant, nous sommes dans un cadre où les réseaux sont constitués pour une courte durée et sont de taille relativement petite (puisqu'ils sont formés par des utilisateurs situés dans une zone géographique de taille réduite). La diffusion de la date à tous les éléments de l'arbre devrait donc être rapide et empêcher des dérives trop importantes entre le haut et le bas de l'arbre. La date étant échangée par les messages de découverte des voisins, il faut que ces opérations soient suffisamment fréquentes, pour permettre que tous les nœuds soient mis à jour au plus vite, mais tout en demeurant raisonnables en terme de trafic. Quoiqu'il en soit, on ne peut pourtant pas exclure la possibilité de l'attaque décrite ci-dessus. Nous considérons cependant acceptable qu'un nœud ait accès à des messages émis à une date antérieur mais très proche de celle à laquelle il est entré dans le groupe. Pour rendre plus improbable encore le type de scénarios décrit ci-dessus, nous faisons en sorte que les utilisateurs n'aient pas accès à la date (elle n'est échangée que par des messages chiffrés et non signés), ni à l'identité du *leader* (dans l'interface graphique, l'affichage du *leader* ne sert qu'à voir l'efficacité de l'algorithme). Sans ces informations, il sera impossible de connaître les zones de dates plus anciennes et l'attaque ne pourra réussir que par hasard.

[1] puisqu'il les voit passer même s'il ne peut pas les lire.

3.3 Modélisation de l'algorithme d'élection de leader proposé

Les graphes et les systèmes de réécriture offrent un cadre efficace et puissant pour la visualisation et l'implémentation d'algorithmes distribués. Ils permettent de les représenter très simplement et très intuitivement. Ces outils sont présentés dans [100, 101, 109].

3.3.1 Introduction aux outils de modélisation

Notions de graphes.

Conformément aux définitions du premier chapitre de ce document, chaque ensemble dispositif - utilisateur est représenté par un nœud. Graphiquement, un nœud (appelé aussi sommet) est représenté par un cercle contenant une lettre représentant son identité. L'interconnexion de deux nœuds, c'est-à-dire la possibilité pour eux de communiquer (bilatéralement), est représentée par des traits pleins (arêtes). Des étiquettes peuvent être associées aux nœuds et/ou aux arêtes. Ces étiquettes représentent l'état du nœud/de l'arête à un moment donné de l'algorithme. Ce peut être une lettre, un chiffre, un couple d'éléments ou toute autre combinaison définie sur un alphabet. L'évolution d'un graphe est représentée par la perte ou l'apparition d'arêtes, le déplacement de nœuds (pour les réseaux mobiles) ou le changement d'état des étiquettes.

Fig. 47 – Représentation des graphes pour l'application SManet

3.3.2 Règles de réécriture

L'évolution d'un graphe suit des traitements déterminées par un algorithme. Chaque opération de l'algorithme peut être représentée graphiquement par des règles de réécriture composées de trois éléments :

- une expression gauche : elle représente la configuration d'un ensemble de nœud et d'arêtes à un moment donné ;
- une expression droite : elle représente un nouvel état des nœuds et des arêtes de l'expression gauche après application d'une règle ;
- une flêche qui va de la gauche vers la droite pour exprimer la transformation d'une configuration en une autre.

Dans notre cas, dans une règle seul un nœud peut changer d'état à la fois ce qui nous permet de retranscrire plus aisément les règles de réécriture en code car chaque opération est ainsi détaillée par une règle.

Preuves. Les systèmes de réécriture offrent aussi un moyen relativement simple de prouver et valider un algorithme. Dans [101], il est prouvé que si le système de réécriture est correct alors l'algorithme correspondant l'est aussi. Or, démontrer la validité d'un système de réécriture est nettement plus simple que de prouver directement celle de l'algorithme.

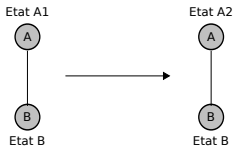


Fig. 48 – Modèle des règles de réécriture utilisées

Dans les travaux cités, une méthodologie est donnée pour une démonstration simple de la validité d'un système de réécriture.

Cadre mobile. A la base, ces systèmes de réécriture ont été développés pour des réseaux fixes. Ils ont ensuite évolué pour s'adapter à la mobilité. Un modèle, les DA-GRS (*Dynamicity-aware Graph Relabeling System*) a été développé. Il a été introduit dans [39], puis utilisé dans [38] pour décrire un algorithme de forêt d'arbres couvrants qui se maintient de manière autonome au fil des évolutions topologiques du réseau.

3.3.3 Application à l'algorithme d'élection de *leader* proposé

a) Définition des notations utilisées

Nous présentons ici les notations relatives aux graphes que nous allons utiliser (fig. 49) :
- G : graphe non nécessairement connexe, éventuellement composé de plusieurs sous-graphes connexes représentant chacun un îlot ;
- $V(G)$: ensemble des sommets d'un graphe ;
- $N(i)$: ensemble des voisins d'un nœud i ;
- $S(i)$: ensemble des fils d'un nœud i avec $S(i) = \{k \in V(I)/P_k = i\}$ où P_X représente le prédécesseur du nœud d'identité X. Intuitivement, les fils d'un nœud sont ceux qui passent par lui pour atteindre le *leader* par le plus court chemin. Ainsi le nœud i est appelé prédécesseur ou père de chacun de ses fils. Chaque nœud n'a qu'un seul père puisqu'il y a un unique plus court chemin.
- Br_i, l'ensemble des nœuds frères du nœud i, c'est-à-dire les nœuds de même prédécesseur que i (P_i).

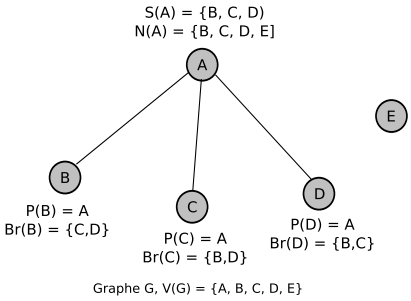


Fig. 49 – Elément d'un graphe

Chaque nœud du graphe a pour étiquette un quadruplet Q(i) = (id_i, lid_i, n_i, P_i) avec :
- id_i, l'identité du nœud ($id_i \in L$) ;

- l_i, l'identité du *leader* de l'îlot dans lequel se trouve le nœud ($l_i \in L$) ;
- n_i le nombre de sauts minimum jusqu'au *leader* ($n_i \in \mathbb{Z}$).
  - si $n_i = -1$ on parle d'état indéfini : le nœud n'a plus de chemin vers son *leader* et il a donc lancé une recherche d'un nouveau chemin auprès de ses fils ;
  - si $n_i = 0$ on parle d'état nul : le nœud n'a plus de chemin vers son *leader* et n'en a pas trouvé de nouveau parmi ses éventuels fils.
- P_i, le premier nœud du chemin le plus court amenant au *leader*, le prédécesseur.
  - si $P_i = 0$, le nœud a détecté une rupture et lancé une recherche parmi ses voisins ;
  - si $P_i = id_i$, le nœud est lui-même un *leader*.

b) Les règles de réécriture de l'algorithme

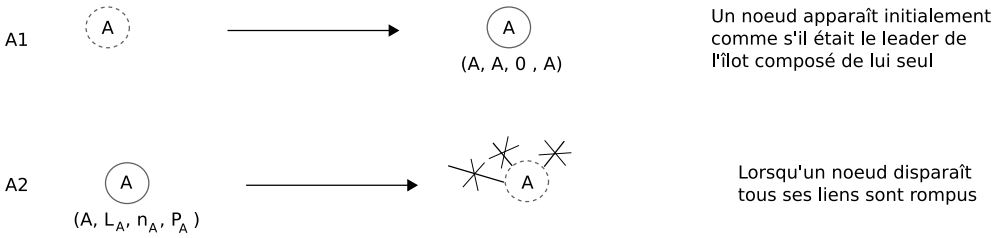
Nos réécritures respectent les règles suivantes afin de faciliter leur compréhension :
- le nœud A est le nœud d'où provient le changement ;
- le nœud B est le nœud sur lequel s'exécute l'algorithme (un seul nœud peut changer d'état dans une règle) ;
- un nœud N est un nœud voisin quelconque ;
- si un nœud est dans l'état indéfini ou nul on l'indique (respectivement -1 ou 0), sinon on écrit n_i ;
- si A est le prédécesseur de B on l'indique, sinon on écrit P_B (un prédécesseur A est explicitement précisé).
- il n'y a pas de règles pour les échanges n'entraînant aucun changement d'état de B sauf pour les règles de connexion pour représenter l'apparition d'une arête.

De plus, à chaque étape d'exécution, notre algorithme impose le respect des invariants suivants :
- si deux nœuds de *leaders* différents se connectent alors le nœud de *leader* le plus grand (dit perdant) adopte le *leader* le plus petit (dit gagnant) ;
- à la réception d'un message informant d'une rupture du chemin vers leur leader, les nœuds ne considèrent pas les chemins provenant de leurs éventuels frères comme de nouveaux chemins pour eux car il se peut que ces frères n'aient pas encore reçu le message indiquant la rupture, on ne peut donc pas prendre le risque de les adopter alors qu'ils sont en rupture sans encore le savoir ;
- si un nœud est défini son prédécesseur est forcément défini ;
- un nœud peut être dans l'état que nous avons appelé indéfini uniquement si son prédécesseur est indéfini lui-même ou si son prédécesseur est nul parce que c'est celui qui a détecté la rupture.
- un nœud peut être dans l'état que nous avons appelé nul :
  - si c'est le leader (son prédécesseur est égal à lui-même) ;
  - s'il a détecté une rupture, a des fils et est en recherche d'un nouveau chemin auprès de ses fils ;
  - si son prédécesseur est dans l'état indéfini ou nul et si aucun de ses fils n'est dans l'état indéfini.

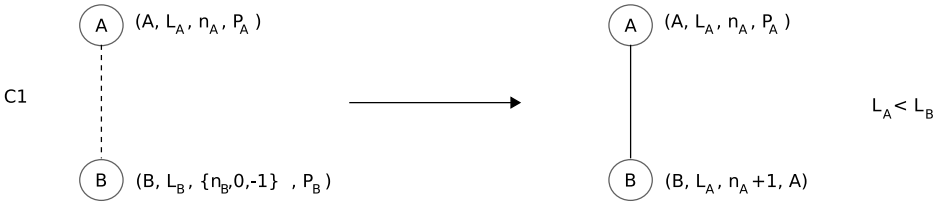
Les règles de réécriture sont classées de la manière suivante :
- les règles A_x sont des règles d'apparition de nœuds ;
- les règles C_x sont des règles de connexion ;
- les règles M_x sont des règles de mise à jour du quadruplet d'un nœud ;
- les règles D_x sont des règles en lien avec une déconnexion.

Règles d'apparition et de disparition des nœuds

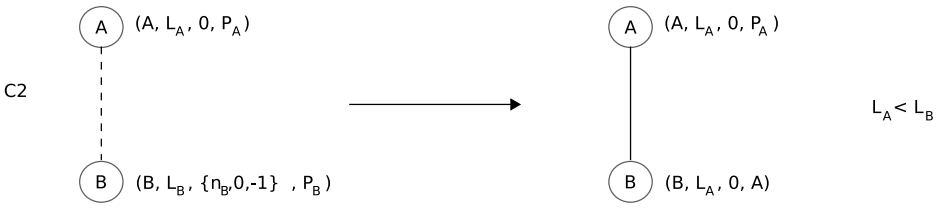
Remarque : ces règles sont internes au modèle et ne sont pas des opérations exécutées par l'algorithme. Elles servent à modéliser l'aspect dynamique du réseau et des entités qui le composent.



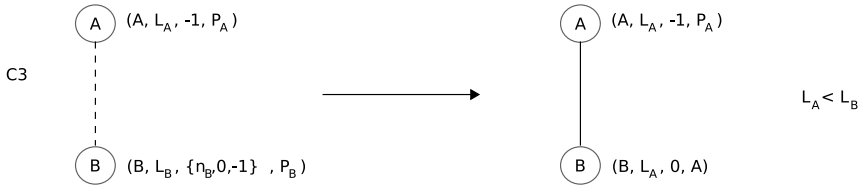
Règles de connexion des nœuds d'îlots différents



Contexte : A offre un leader plus petit à B.
Action : B adopte le leader de A et prend A comme prédécesseur.

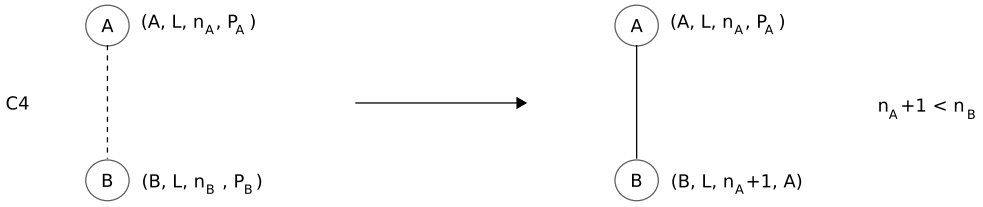


B adopte le leader de A qui est le plus petit malgré l'absence de chemin pour y accéder.

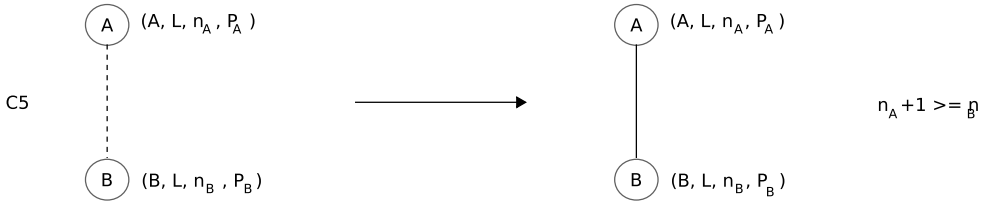


Contexte : A offre un leader plus petit à B.
Action : B adopte le leader de A et prend A comme prédécesseur. Il se met à l'état nul car A n'offre pas de chemin vers son leader (l'adoption est tout de même obligatoire).

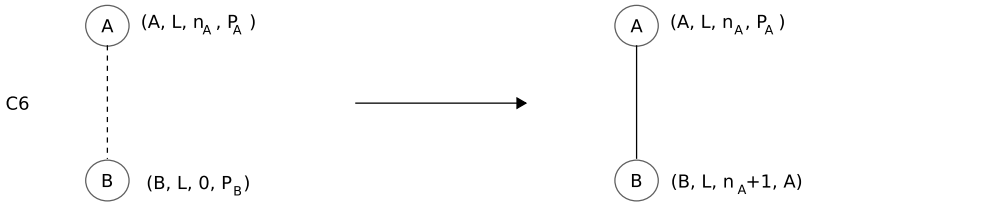
Règles de connexion entre des nœuds d'un même îlot



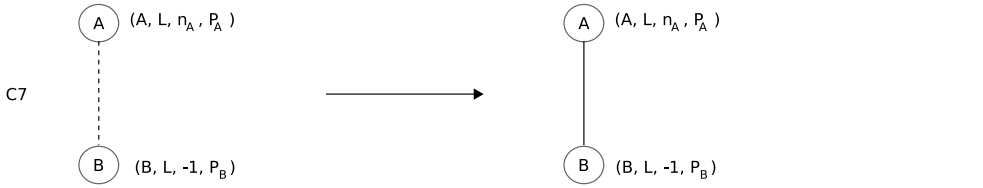
Contexte : A offre un chemin plus petit à B dans le même îlot.
Action : B adopte le leader de A et prend A comme prédécesseur.



Contexte : A n'offre pas de chemin plus petit à B dans le même îlot.
Action : B conserve son chemin.

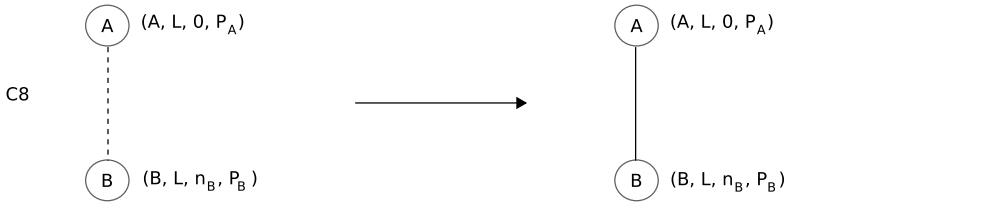


Contexte : B n'a plus de chemin vers le leader ; A offre un chemin vers le leader
Action : B prend A comme prédécesseur.

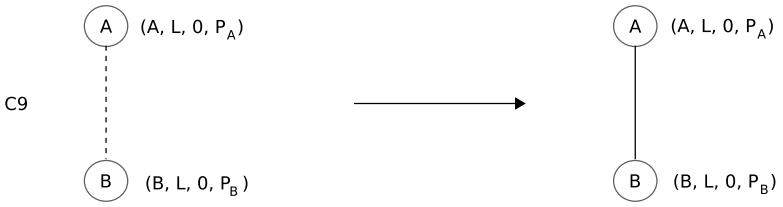


Contexte : B est en recherche d'un nouveau chemin ; A offre un chemin mais B a toujours des fils en recherche.
Action : B ne change pas, il doit attendre que tous ses fils aient fini leur recherche avant de pouvoir éventuellement adopter A comme prédécesseur.

.



Contexte : B a un chemin vers le leader ; A n'a pas de chemin vers le leader.
Action : B conserve son état.



Contexte : B n'a plus de chemin vers le leader ; A n'en offre pas.
Action : B ne change pas d'état.

Contexte : B est en recherche d'un chemin ; A n'offre pas de chemin vers le leader.
Action : B ne change pas d'état.

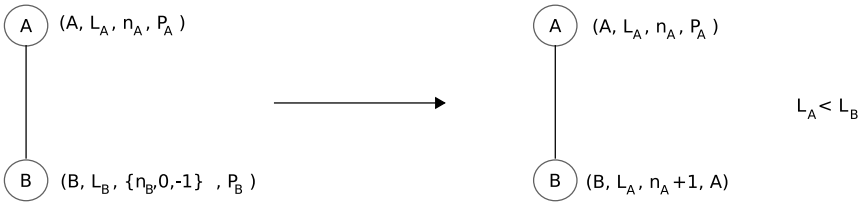
Contexte : B a un chemin vers le leader ; A n'offre pas de nouveau chemin.
Action : B ne change pas d'état.

Contexte : B n'a plus de chemin ; A n'offre pas de nouveau chemin (il est en recherche).
Action : B ne change pas d'état.

Contexte : B est en recherche d'un chemin ; A est aussi en recherche de chemin.
Action : B ne change pas d'état.

Règles de mise à jour

M1

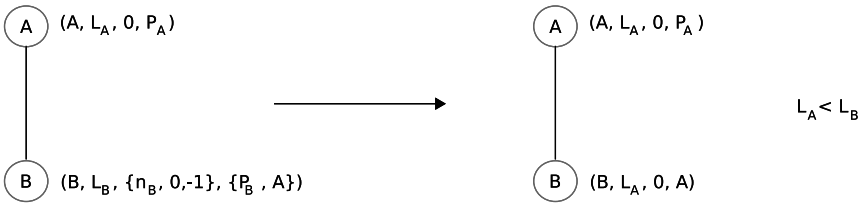


Contexte : A est un voisin qui a adopté un nouveau leader plus petit.
Action : B adopte le nouveau leader et prend A comme prédécesseur.

M2

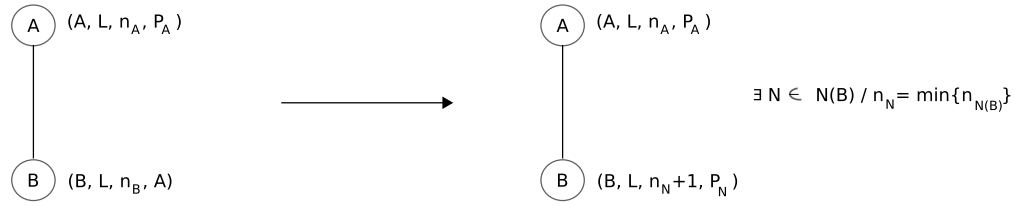
Contexte : le prédécesseur de B a adopté un nouveau leader plus petit.
Action : B adopte le nouveau leader et garde A comme prédécesseur.

M3



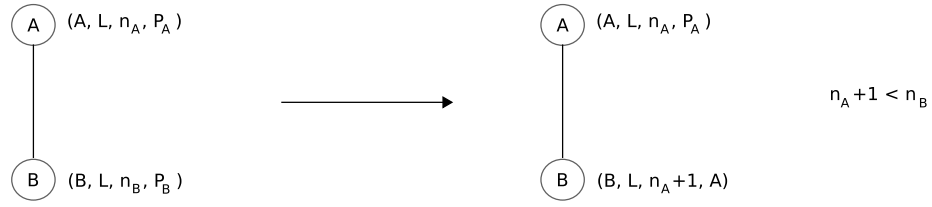
Contexte : un voisin (qui peut être le prédécesseur) a adopté un leader plus petit.
Action : B adopte le nouveau leader, prend A comme prédécesseur et se met à null puisque A n'a pas de chemin défini (l'adoption du leader le plus petit est obligatoire, voir les invariants imposés p99).

M4



Contexte : le chemin de A a changé, il n'est plus forcément le plus court chemin vers le leader.
Action : B prend comme prédécesseur le voisin offrant le plus petit chemin (ça peut être A).

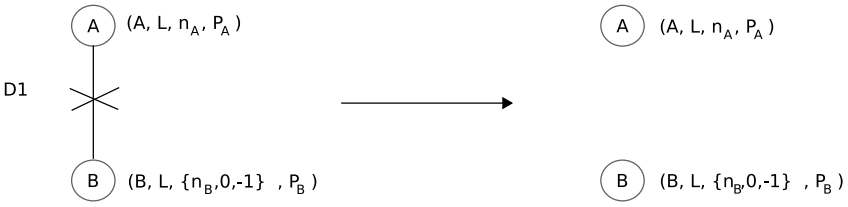
M5



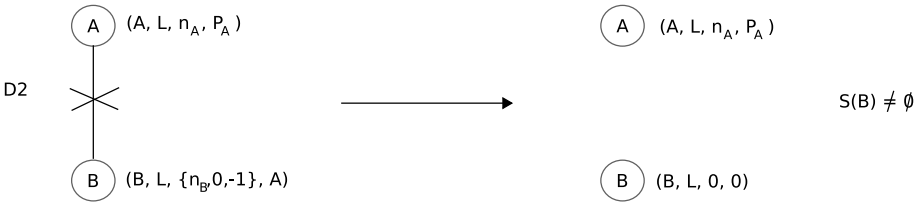
Contexte : B a un chemin ; A offre un plus petit chemin que P_B.
Action : B prend A comme nouveau prédécesseur.

Contexte : B n'a plus de chemin vers son leader ; un voisin A (pouvant être le prédécesseur) a trouvé un chemin.
Action : B met à jour son chemin et garde A comme prédécesseur.

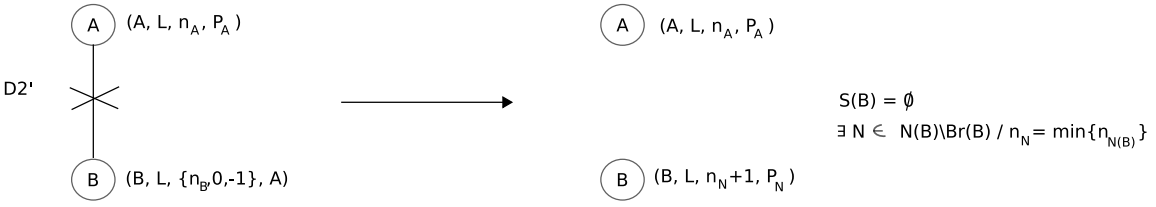
Règles de déconnexion des nœuds



Contexte : B a détecté une rupture d'un voisin qui n'est pas son prédécesseur (P_B différent de A).
Action : B ne change pas d'état.

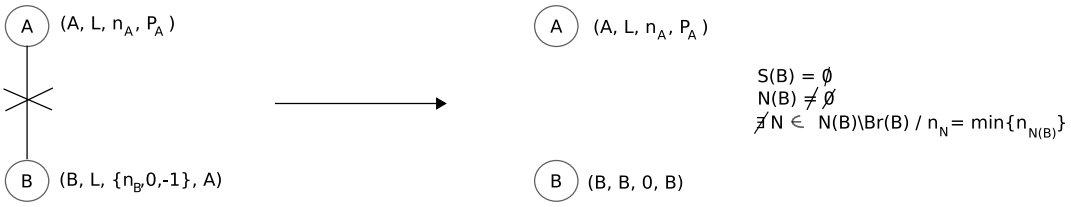


Contexte : B a détecté une rupture du lien avec son prédécesseur ; il a des fils.
Action : B se met en recherche d'un nouveau chemin.



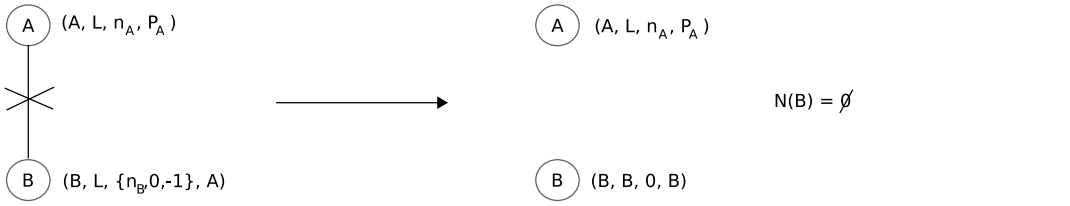
Contexte : B a détecté une rupture du lien avec son prédécesseur ; il n'a pas de fils ; il a des voisins autres offrant un plus court chemin (on ne sait pas si les frères ont déjà reçu le message indiquant la rupture).
Action : B adopte comme prédécesseur le voisin qui offre le plus petit chemin.

D2''



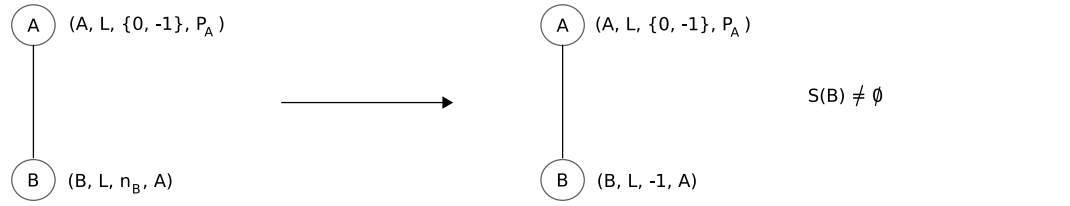
Contexte : B a détecté une rupture du lien avec son prédécesseur ; il n'a pas de fils ; il n'a pas de voisins autre que des frères offrant un chemin (on ne sait pas si les frères ont déjà reçu le message indiquant la rupture).
Action : B devient son propre leader.

D3



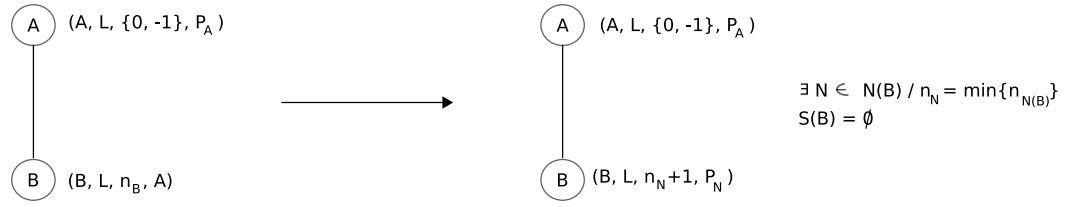
Contexte : B a détecté une rupture du lien avec son prédécesseur ; il n'a plus de voisin.
Action : B devient son propre leader.

D4



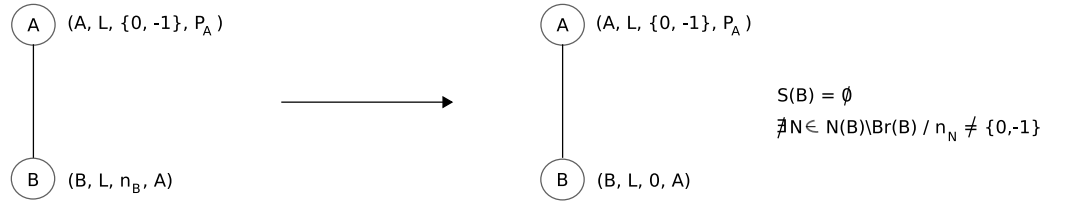
Contexte : le prédécesseur de B a lancé une recherche d'un nouveau chemin auprès de ses fils.
Action : B lance lui-même une recherche auprès de ses fils.

D5



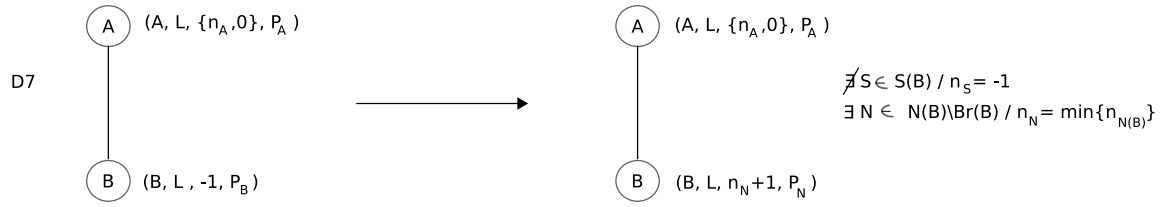
Contexte : le prédécesseur de B est en recherche d'un nouveau chemin ; B n'a pas de fils mais il a des voisins qui offrent un chemin (A est en recherche et à l'état nul si c'est lui qui a détecté la rupture).
Action : B prend comme prédécesseur le voisin offrant le plus court chemin.

D6



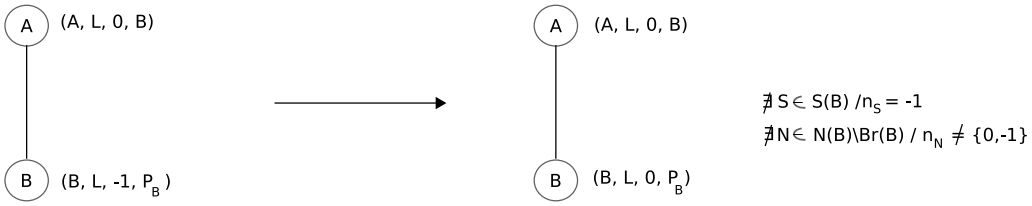
Contexte : le prédécesseur de B est en recherche d'un nouveau chemin et A est non défini ; B n'a pas de fils ni aucun voisin (autre qu'un frère) qui lui offre un chemin.
Action : B se met à l'état nul (indiquant l'absence de chemin).

D7



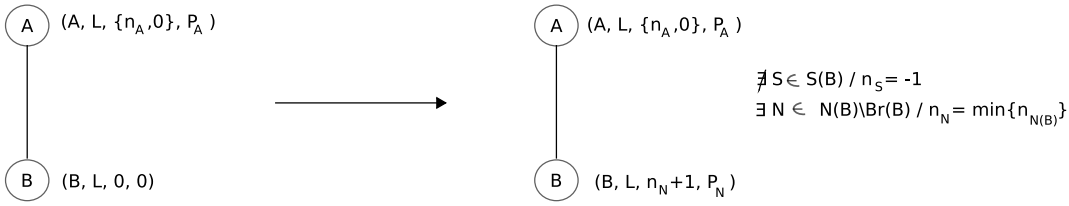
Contexte : B est en recherche d'un chemin ; A était le dernier fils de B encore en recherche d'un nouveau chemin ; au moins un voisin autre qu'un frère offre un chemin vers le leader (on ne sait pas si les frères ont déjà reçu le message indiquant la rupture).
Action : B prend comme prédécesseur le voisin offrant le plus court chemin.

D8



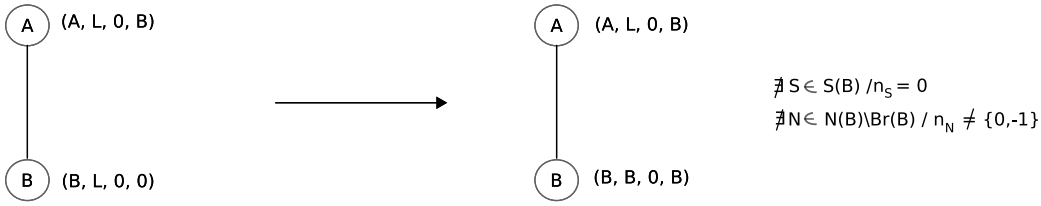
Contexte : B est en recherche d'un chemin ; A était le dernier de ses fils à être encore en recherche : aucun voisin autre qu'un frère n'offre de chemin vers le leader (on ne sait pas si les frères ont déjà reçu le message indiquant la rupture).
Action : B se met à l'état nul (qui indique l'absence de chemin).

D9



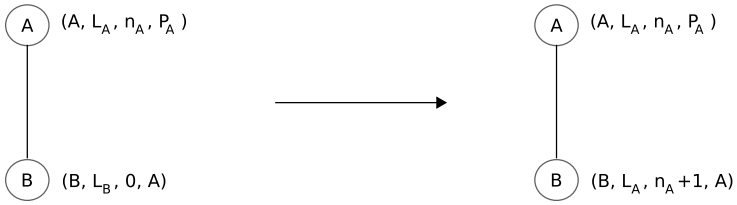
Contexte : B avait détecté une rupture du lien avec son prédécesseur ; A était son dernier fils encore en recherche ; B a au moins un voisin autre qu'un frère lui offrant un chemin vers le leader (on ne sait pas si les frères ont déjà reçu le message indiquant la rupture).
Action : B prend comme leader le voisin lui offrant le plus court chemin.

D10



Contexte : B avait détecté une rupture sur le lien avec son prédécesseur ; A était le dernier de ses fils encore en recherche d'un chemin ; aucun voisin autre qu'un frère n'offre de chemin vers le leader (on ne sait pas si les frères ont déjà reçu le message indiquant la rupture).
Action : B devient son propre leader.

D11



Contexte : B n'a plus de chemin vers le leader ; son prédécesseur a adopté un nouveau leader.
Action : B adopte le nouveau leader (même s'il n'est pas plus petit) et garde A comme prédécesseur.

3.3.4 Preuves

La preuve de ce système de réécriture est donnée en Annexe A.

3.3.5 Etudes des différents cas de figure

Pour vérifier que tous les cas de figure sont traités et qu'ils sont valides, nous les avons représentés dans les tableaux ci dessous. Remarque : on ne se préoccupe ici que de l'élément plus court chemin, les autres modifications des autres éléments de l'étiquette ayant des changements très simples.

Règles de connexions entre deux nœuds de *leader* différents.

Ri	A	B	Conditions	Explications	Contexte
C1	n_A	n_B,0,-1 -> n_A+1		Le chemin de A est défini donc B l'adopte	Pour B, A est le seul lien avec l'îlot du *leader* gagnant (sinon il aurait déjà adopté le *leader* puisque qu'une connexion de deux nœuds de *leader* différent entraîne forcément l'adoption du nœud de *leader* gagnant par le nœud de *leader* perdant.
C2	0	n_B,0,-1 -> 0		A n'a pas de chemin défini donc B non plus puisque ses voisins et ses fils sont de l'îlot du *leader* perdant.	
C3	-1	n_B,0,-1 -> 0		A ne sait pas s'il a un chemin vers le *leader* donc B se met à *nul* puisque ses voisins et ses fils sont d'un autre îlot.	

Règles de connexion entre des nœuds d'un même îlot.

Ri	A	B	Conditions	Explications	Contexte
C4	n_A	n_B -> n_A+1	A offre un plus petit chemin	B adopte A	A chaque nouvelle connexion entre des nœuds d'un même îlot il suffit de vérifier si un chemin plus petit est proposé.
C5		n_B	A n'offre pas de plus petit chemin	Aucun changement	
C6		0 -> n_A+1	A offre forcément un plus petit chemin	B adopte A comme prédécesseur car A lui offre un chemin alors que lui-même n'en a plus	
C7		-1 -> -1	A offre forcément un plus petit chemin (le seul défini)	B doit attendre que tout ces fils aient fini leur recherche (B à 0 ci-dessus	
C8	0	n_B -> n_B		Aucun changement : A n'offre pas de chemin plus petit	Si A est indéfini ou nul, une connexion à ce nœud n'entraine aucune modification puisqu'il n'offre pas de nouveau chemin.
C9		0 -> 0		Aucun changement : A n'offre pas de nouveau chemin	
C10		-1 -> -1		Aucun changement : A n'offre pas de nouveau chemin	
C11	-1	n_B -> n_B		Aucun changement : A n'offre pas de chemin plus petit	
C12		0 -> 0		Aucun changement : A n'offre pas de nouveau chemin	
C13		-1 -> -1		Aucun changement : A n'offre pas de nouveau chemin	

Règles de mise à jour entre des nœuds de leader différents.

Ri	A	B	Conditions	Explications	Contexte
M2	n_A	n_B,-1 -> n_A+1	A prédécesseur	B adopte le *leader* de A et garde A comme prédécesseur dans le nouvel îlot	Pour B, A est le seul lien avec l'îlot du leader gagnant (sinon il aurait déjà adopté le leader puisque qu'une connexion de deux nœuds de *leader* différent entraîne forcément l'adoption du nœud de *leader* gagnant par le nœud de *leader* perdant). Il faut regarder si A était le prédécesseur ou non à cause du cas particulier où B est nul.
D11		0 -> n_A+1	A prédécesseur	B ne peut être nul que si A était à indéfini ou nul. A peut être redevenu défini et d'un autre leader si : – il s'est relié à un autre îlot avec un leader gagnant => il faut donc l'adopter ; – un îlot a été détecté et un nœud vient de s'auto-proclamer leader => il faut l'adopter même si le leader proposé n'est pas le plus petit.	
M1		n_B,0,-1 -> n_A+1	A non prédécesseur	B adopte A puisque c'est son seul chemin vers son nouveau leader.	
M3	0	n_B,0,-1 -> 0	A prédécesseur ou non	B adopte le leader de A, prend A comme prédécesseur et se met à nul puisqu'il n'y a aucun chemin défini (ses fils et ses autres voisins sont de son ancien îlot).	
	-1	n_B,0,-1	A prédécesseur ou non	Impossible : un nœud n'adopte jamais un nouvel îlot en mettant son chemin à indéfini (il peut être nul comme ci-dessus). Les voisins de B ne peuvent donc pas être à indéfini (possible que par une connexion).	

Règles de mises à jour entre des nœuds d'un même îlot.

Ri	A	B	Conditions	Explications	Contexte
M4	n_A	n_B -> n_{min}+1	A prédécesseur	B prend comme prédécesseur le nœud offrant le plus petit chemin.	Un voisin a été modifié et est défini. Il faut regarder : – si c'est le prédécesseur, offre-t-il un chemin plus intéressant ? – si ce n'est pas le prédécesseur : – si B est défini, est ce qu'il offre un chemin plus petit ? – sinon si B est nul, l'est-il parce qu'il a détecté une rupture ou parce que tous ces fils sont à nuls ? – sinon si B est indéfinis, lui reste-t- il des fils à indéfinis ?
M5		n_B -> n_A+1	A non prédécesseur ; A offre un plus petit chemin	B prend comme prédécesseur A qui a le plus court chemin	
		n_B -> n_B	A a un plus grand chemin	Aucun changement : B a déjà le plus court chemin.	
M6		0 -> n_A+1	A prédécesseur	Le prédécesseur a trouvé un chemin donc il l'adopte	
D9		0 -> n_{min}+1	A non prédécesseur ; B a détecté la rupture	A est le dernier fils de B à répondre. A va choisir parmis les fils définis (qui ont trouvé un chemin) celui qui offre le plus petit chemin	
M6		0 -> n_A+1	A non prédécesseur ; B n'a pas détecté la rupture	B n'a plus de chemin et A lui en propose un donc il le prend comme prédécesseur	
		-1	A prédécesseur	Impossible car le prédécesseur ne peut pas changer tant qu'il reste des fils à indéfinis (B)	
D7		-1 -> n_{min}+1	A non prédécesseur ; Il n'y a plus de fils à -1	A est le dernier fils à répondre sinon B serait déjà à 0. B n'a plus de chemin et A lui en propose un donc il l'adopte	
		-1 -> -1	A non prédécesseur ; Il y a encore des fils à -1	Aucun changement : B ne peut prendre A comme prédécesseur que si tous ces fils sont à 0.	

Ri	A	B	Conditions	Explications	Contexte
D4	0	n_B -> -1	A prédécesseur ; Il y a des fils	B lance une recherche	Un voisin est devenu nul, et B définis, il faut regarder : – si A était le prédécesseur alors c'est qu'il vient de détecter une rupture (c'est le seul cas où un prédécesseur est à 0 sans que ces fils le soient tous aussi) et n'a plus de chemin vers son leader. Il faut alors voir : – si B a des fils, lancer une recherche – sinon s'il existe un chemin dans le reste de son voisinage
D5		n_B -> n_{min}+1	A prédécesseur ; Il n'y a pas de fils ; Il existe un chemin	B adopte le seul chemin possible	
D6		n_B -> 0	A prédécesseur ; Il n'y a pas de fils ; Il n'existe pas de chemin	B n'a plus aucun chemin vers le *leader* et c'est une feuille, il se met à nul	
		n_B -> n_B	A non prédécesseur	Aucun changement : A n'offre pas de nouveau chemin	
		0 -> 0	A prédécesseur	Aucun changement : A était déjà à nul car un nœud ne peut être à nul que si son prédécesseur est à déjà à indéfini ou nul	Un voisin est devenu nul, et B est à 0. Il faut regarder : – si A était le prédécesseur – sinon – si B est à 0 car il a détecté une rupture, est-ce que tous ces fils ont finis leur recherche et si oui est-ce qu'il existe un chemin ; – sinon tous ces fils sont donc à nul
		0 -> 0	A non prédécesseur ; B a détecté la rupture ; Il reste des fils indéfinis	Aucun changement : B doit attendre que tous ces fils aient fini leur recherche	
D9		0 -> n_{min}+1	A non prédécesseur ; B a détecté la rupture ; Il n'y a plus de fils indéfinis ; Il existe un chemin	B adopte le chemin le plus court parmis ces voisins (fils défini y compris)	
D10		0 -> 0	A non prédécesseur ; B a détecté la rupture ; Il n'y a plus de fils indéfinis ; Il n'existe pas de chemin	B devient le leader (son chemin reste nul mais son leader change)	
		0 -> 0	A non prédécesseur ; B n'a pas détecté la rupture	Aucun changement : A n'offre pas de nouveau chemin	
		-1 -> -1	A prédécesseur	Aucun changement : A ne peut être à 0 avec des nœud indéfinis que si c'est lui qui a détecté la rupture. A était donc déjà à nul	Un voisin est devenu nul, et B est à -1. Il faut regarder : – si A était le prédécesseur (uniquement possible si A a détecté une rupture) – sinon – s'il reste des fils indéfinis – sinon s'il existe un chemin
		-1 -> -1	A non prédécesseur ; Il reste des fils indéfinis	Aucun changement : B attend que tous ces fils aient fini leur recherche	
D7		-1 -> n_{min}+1	A non prédécesseur ; Il n'y a plus de fils indéfinis ; Il existe un chemin	B adopte le plus court chemin puisque ces fils ne lui en offre aucun	
D8		-1 -> 0	A non prédécesseur ; Il n'y a plus de fils indéfinis ; Il n'existe pas de chemin	B se met à nul car ni ces fils ni son voisinage ne lui offrent de chemin	
D4	-1	n_B -> -1	A prédécesseur ; Il y a des fils	A informe ces fils qu'il n'a plus de plus court chemin vers le leader	A informe B qu'il n'a plus de chemin vers le leader. Il faut regarder : – si B a des fils (lancer la recherche) – s'il n'a pas de fils – s'il existe un chemin ailleurs.
D5		n_B -> n_{min}+1	A prédécesseur ; Il n'y a pas de fils ; Il existe un chemin	B adopte le seul chemin existant	
D6		n_B -> 0	A prédécesseur ; Il n'y a pas de fils ; Il n'existe pas de chemin	B est une feuille et se met donc à nul	
		n_B -> n_B	A non prédécesseur	Aucun changement : A n'offre pas de nouveau chemin	
		0 -> 0	A prédécesseur	Aucun changement : A était déjà à -1 (un fils ne passe à indéfini ou nul que quand son prédécesseur dévient indéfini)	

Ri	A	B	Conditions	Explications	Contexte
	-1	0 -> 0	A non prédécesseur	Aucun changement : A n'offre pas de chemin	
		-1 -> -1	A prédécesseur	Aucun changement : A était déjà à -1 (un fils ne passe à indéfini ou nul que quand son prédécesseur devient indéfini)	
		-1 -> -1	A non prédécesseur	Aucun changement : A n'offre pas de nouveau chemin	

3.3.6 Conclusion

L'algorithme d'élection d'un *leader* que nous proposons est adapté aux MANets purs. Il n'impose aucune contrainte aux utilisateurs et les nœuds ont des tâches équitables. Le développement de cet algorithme nous a permi d'utiliser les deux briques de sécurité que nous avons mises en place dans le chapitre précédent : la gestion des identités sûres et l'établissement de groupes sécurisés.

Chapitre 4

SManet : Implémentation de l'application

Sommaire

Ce chapitre a pour but de présenter l'application SManet qui implémente notre solution de sécurité. Dans la première section nous présentons les outils matériels et logiciels que nous avons utilisés. La deuxième section est consacrée à la description détaillée de notre implémentation. Enfin, dans la dernière section, nous en donnons la validation formelle ainsi que les résultats de tests pratiques.

4.1 Outils utilisés pour effectuer l'implémentation

Dans cette section nous présentons l'ensemble des outils logiciels utilisés pour implémenter notre application. Nous avons utilisé deux langages différents sous deux plateformes différentes : C# avec *Visual Studio .NET* pour l'implémentation de la partie utilisateur et *Java Card* sous *Eclipse* pour la partie carte à puce.

4.1.1 Visual Studio 2005 et le .NETCF

Visual Studio 2005 [154] est un logiciel de développement sous *Windows* permettant de générer divers types d'applications (pour le web avec ASP, pour la portabilité avec .NET, XML, les applications *windows* classiques, etc.) dont celles qui nous intéressent, les applications mobiles pour PDAs. Nous avons utilisé l'extension *Windows Mobile 5.0 SDK for Pocket PC*. Le *framework* .NETCF [117] (*.NET Compact Framework*), extension du *framework* .NET [141], offre en outre des outils facilitant le développement d'applications destinées aux terminaux mobiles. Plus précisément, il apporte l'harmonisation des APIs et l'interopérabilité entre plusieurs langages : le code est développé dans un langage (supporté) quelconque, puis il est compilé par un compilateur spécifique au langage en un code intermédiaire MSIL (*Microsoft Intermediate Language*) indépendant du matériel et du système d'exploitation. Enfin, la compilation en code natif est effectuée par le JIT (Just In Time) Compiler lors de la première exécution par le CLR (*Common Language Runtime*).

Pour installer et exécuter des programmes sur un PDA, il faut tout d'abord charger un fichier .cab particulier, adapté à la version de *Windows* supportée par ce PDA. Ensuite, chaque application doit passer par un certain nombre d'étapes pour pouvoir être elle-même chargée sur le PDA (cf. 4.1.5(c) page 114).
Pour résumer, on peut dire que la plateforme *visual studio 2005* et le *framework* .NETCF permettent le chargement et l'installation d'une application et de tous les éléments nécessaires (fichiers .cab, bibliothèques, etc.) pour la rendre opérationnelle sur un dispositif mobile simulé ou réel.

4.1.2 Le langage C#

Le langage C# [141] (standard ECMA) est un langage de programmation objet basé sur l'architecture .NET. C'est une synthèse des langages C, C++ et Java, avec des fonctionnalités améliorées. Il s'intègre à .NET dont il partage les formats d'objet. Un des buts majeurs de ce langage est de proposer une meilleure gestion de la sécurité des applications. Ses apports principaux sont les suivants :

– les délégués : se sont des équivalents des pointeurs de fonctions (inexistants en Java) ;

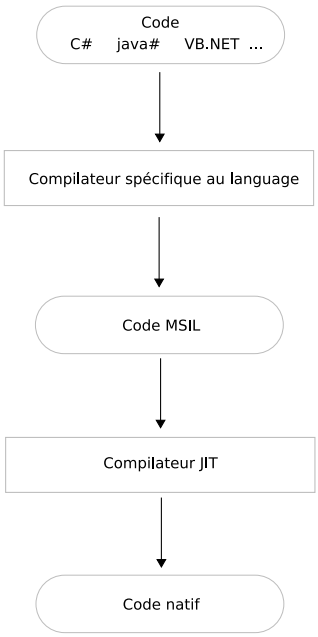


Fig. 50 – Schéma de compilation/exécution du *framework* .NET

- les propriétés : se sont des méthodes spéciales utilisées pour faciliter l'accès aux champs privés d'une classe ;
- les attributs : ils permettent d'insérer des méta-données dans une déclaration (classe, propriété, méthode, etc.) puis de les récupérer par un système de « réflexion » (cf. chapitre 7 dans [141]) ;
- la manipulation des pointeurs : ceci est autorisé uniquement dans des zones marquées *unsafe* ayant des propriétés particulières de sécurité ;
- les fichiers d'en-tête : ils sont remplacés par un système d'espace de noms ;
- nature des fichiers : un même fichier peut contenir n'importe quel nombre de classes, structures, interfaces et événements.

Nous avons choisi le langage C# car outre son haut niveau offrant des facilités de développement, il est utilisable avec la bibliothèque *SpringCard* [142] (supportant aussi C++, VB, permettant les communications avec le lecteur de carte à puce que nous utilisons (*Pro-Active SpringCard-CF*). Cette bibliothèque a été développée dans le cadre du projet MUSCLE (*Mouvement For The Use of Smart Cards in a Linux Environment*) [112].

4.1.3 Eclipse et JCOP (*Java Card OpenPlatform*)

Eclipse [55] est un environnement de développement dont les fonctionnalités sont développées sous forme de *plug-in*, ce qui permet en théorie de créer des applications sous n'importe quel langage de programmation.
JCOP (*Java Card OpenPlatform* [87]) est le système d'exploitation installé sur les cartes à puce *Philips* que nous utilisons. Il existe un *plug-in* JCOP pour Eclipse qui comporte un environnement de développement, un *debugger* JCOP et un *shell* pour envoyer des commandes APDU (cf. 4.1.5(a) page 113) aux cartes à puce. Nous avons donc choisi cet

environnement.

4.1.4 Le langage JavaCard

Nous utilisons la version 2.1 du langage JavaCard [44] pour ce qui concerne l'application embarquée sur la carte à puce. Les applications JavaCards sont appelées des *applets*. Par rapport aux autres outils de développement pour programmer ces cartes, le langage JavaCard apporte des facilités car il est composé d'un sous-ensemble de Java et de fonctionnalités qui lui sont propres (fig. 51).

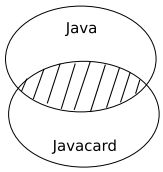


FIG. 51 – Relation entre les langages Java et JavaCard

Il offre aussi l'indépendance du code vis-à-vis du matériel par sa pré-compilation en *bytecode* qui est un code intermédiaire (non natif) pouvant être exécuté sur n'importe quel matériel disposant d'une machine virtuelle JCVM (*Java Card Virtual Machine*, environnement logiciel permettant l'exécution du langage JavaCard indépendemment du matériel).

a) Particularités du langage JavaCard.

Ce langage est très réduit (même par rapport au *J2ME*) mais il possède des mécanismes permettant de prendre en compte les spécificités des cartes à puce. En particulier, la gestion mémoire étant problématique dans ces dispositifs, deux types d'objets sont proposés :

- les objets temporaires, stockés dans la mémoire temporaire (RAM) et qui perdent leur valeur à la fin de la session, fin qui peut être provoquée par le retrait de la carte, une coupure de courant ou la dé-sélection de l'*applet* (cf. 4.1.5(d) page 114).
- les objets persistants, stockés en mémoire persistantes (EEPROM) qui eux survivent aux différentes sessions.

Dans la version 2.1 que nous utilisons, ce langage ne supporte pas :

- le chargement dynamique de classes ;
- les *threads* ;
- la sérialisation ;
- le clônage des objets ;
- la méthodes $finalize$;
- les tableaux à plusieurs dimensions ;
- les types de données suivants : $float$, *double*, *long*, *char* et *string*. Le langage JavaCard ne supporte que les types simples *short*, *byte*, *boolean*.

Lors du développement d'*applet*, un point très important à considérer est l'absence de ramasse-miettes ou de dés-allocation explicite : une zone mémoire allouée à un objet ne sera pas récupérée si celui-ci n'est plus référencé. Il est donc habituel et fortement conseillé

de créer tous les objets lors de l'installation de l'*applet*. Les différentes méthodes ont donc accès à un nombre d'emplacements mémoire fixe et doivent toujours réutiliser les mêmes objets. Ceci garantit qu'il y aura toujours la place nécessaire pour les objets dont on doit se servir. Il est par ailleurs préférable de prévoir à l'avance l'espace nécessaire plutôt que de risquer une saturation de la mémoire, non gérée par l'application. Le seul moyen de récupérer la mémoire utilisée par une *applet* est d'effacer cette *applet*.

Remarque : pour le développement de notre application ces restrictions nous ont contraint à recréer des outils de gestion de chaînes de caractères et de gestion des tableaux (cf. 4.4.3 page 133).

b) JavaCard, un langage sécurisé

Le langage JavaCard est doté de nombreuses sécurités logicielles : un fort contrôle de l'accès aux méthodes et aux variables d'une *applet* (*public*, *protected*, *private*), un typage fort, l'impossibilité de construire des pointeurs qui pourraient être utilisés à des fins malicieuses (espionnage du contenu de la mémoire) et une séparation des *applets* entre elles par un pare-feu.
Un mécanisme de *bac à sable* (politique de sécurité interdisant toute interaction entre des objets d'*applets* différentes) protège les données de chaque *applet*. Cependant, des échanges entre les différentes *applets* restent possibles, sous certaines conditions, et sous le contrôle de la machine virtuelle.

4.1.5 Les applications JavaCard

a) Composants d'une *applet*.

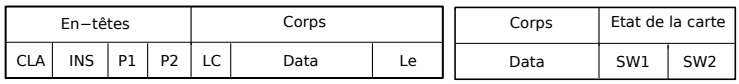
Une *applet* est une application destinée à être exécutée sur une Java Card. Chacune de ces *applet* est identifiée de manière unique par un AID (*Application Identifier*) défini par la norme ISO7816-4. Elle communique avec son terminal hôte au moyen de requêtes passées par des tableaux de *bytes* au format APDU (*Application Protocol Data Unit*, fig. 52). Une requête APDU contient les champs suivants :

- champ CLA : désigne la classe ;
- champ INS : désigne l'instruction (méthode ajoutée par le développeur) à exécuter ;
- champs $P1$ et $P2$: données pour le paramétrage de l'instruction ;
- champ Lc : indique la taille du champs de données envoyées ;
- champ *data* : données envoyées ;
- champ Le : indique la taille des données à envoyer dans la réponse.

Une réponse APDU contient les champs suivants :

- champ *data* : données à renvoyer ;
- champs $SW1$ et $SW2$: indiquent l'état de la carte après l'exécution de la requête à l'origine de cette réponse.

Remarque : les cartes que nous utilisons sont passives, toute requête à la carte provoque une réponse de celle-ci et un message de la carte est obligatoirement le résultat d'une requête de la partie utilisateur.



(a) Requête APDU (b) Réponse APDU

FIG. 52 – Format des APDUs

b) Développement d'une *applet*.

Une *applet* implémente une classe qui étend la classe $javacard.framework.Applet$, celle-ci définissant les méthodes permettant d'interagir avec le JCRE. Ces méthodes sont les suivantes :

- *install*/*uninstall* : la méthode *install* est appelée une fois par le JCRE pour créer une instance de la classe.
- *select*/*deselect* : *select* est une méthode appelée par le JCRE pour sélectionner l'*applet*, c'est-à-dire indiquer à la carte que toutes les requêtes suivantes, jusqu'à l'appel de la méthode *deselect* seront destinées à cette *applet* ;
- *process* : cette méthode est appelée par le JCRE lorsqu'il reçoit une requête adressée à cette *applet* ;
- *register* : cette méthode est appelée dans *install* lorsque tous les paramètres sont initialisés, afin d'enregistrer cette instance d'*applet* dans le JCRE.

Remarque : à la réception d'un APDU, le *Java Card Runtime Environment* (JCRE) invoque l'une de ces méthodes.

c) Installation d'une *applet* (fig. 53).

Le code d'une *applet* est développé dans un environnement Java classique. Ce code est ensuite compilé pour être converti au format *.class*. Les fichiers *.class* sont à leur tour converti au format *.cab* puis chargés dans la carte à puce.

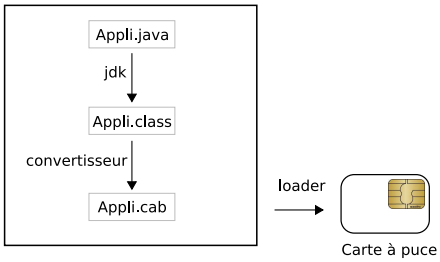


FIG. 53 – Installation d'une *applet*

d) Vie d'une *applet* (fig. 54).

Pour exécuter une *applet* une fois son code chargé dans la carte, il faut tout d'abord en créer une instance avec la méthode *install*. Il faut ensuite sélectionner l'*applet* en invoquant la méthode *select* de celle-ci pour que le JCRE transfère, via sa méthode *process*, les

requêtes APDUs. Ces dernières sont alors traitées par la méthode de l'*applet* indiquée dans leur champs INS.

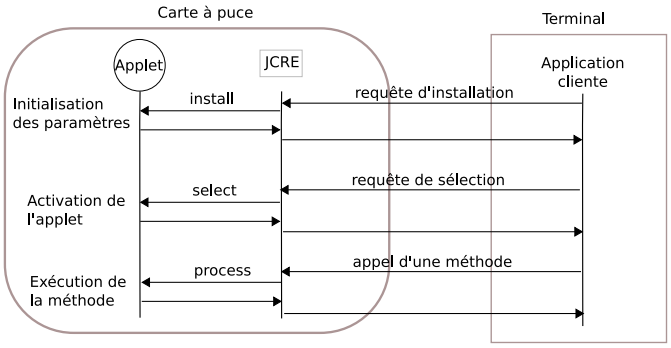


Fig. 54 – Vie d'une *applet*

4.2 Architecture de l'application

Dans cette section nous présentons l'architecture générale de l'application SManet[1], puis, nous verrons les échanges de messages pour chacune de ses fonctionnalités .

4.2.1 Guide de lecture des schémas d'architecture

Nous avons choisi de représenter les échanges mis en jeu pour les différentes fonctionnalités de l'application sur des schémas qui permettent de voir où l'on se situe dans le code. Dans le contexte précis de ce document ceci nous a paru plus pertinent que des diagrammes de séquences classiques qui n'offrent qu'une vue des classes concernées. Sur les différents schémas d'architecture, les échanges de messages entre les instances de classes se font par des canaux représentés par des flèches. Chaque échange est numéroté et correspond à une explication textuelle se trouvant à la suite du schéma. Le contenu d'un message peut varier en fonction du résultat du traitement opéré par la carte à puce, auquel cas les différents contenus possibles sont séparés par le signe « / ».

Les messages échangés entre différentes cartes à puce ont le format de base suivant : *identifiant de requête ; nonce ; données.*

Certaines classes ont un préfixe ayant une signification particulière :
- le préfixe *EC* signifie qu'il s'agit d'un module de communication avec l'extérieur (*External Communication*) ;
- le préfixe *LC* indique un module de communication entre des entités locales (*Local Communication*).
- le préfixe *AC* exprime le lien avec l'*applet* (*Applet Card*) ;
- le préfixe *Mile* (*Mobility adapted Implementation of Leader Election*) est utilisé pour les classes de l'algorithme d'élection du *leader*.

[1] certains modules secondaires n'apparaissent pas sur les schémas pour des raisons de simplification.

4.2.2 Architecture globale

L'implémentation de l'application SManet se compose de cinq modules répartis sur deux niveaux (fig. 55) :

- le niveau utilisateur comprend un module pour l'interface utilisateur, un pour la communication vers l'extérieur et un autre pour les échanges avec la carte à puce ;
- le niveau carte à puce comprend l'*applet* principale, i.e. celle qui gère les identités et les groupes, le module d'élection d'un *leader* (cf. 3.2 page 87) et un module d'outils développés pour permettre la gestion des structures utilisées dans les autres modules de la carte. L'architecture générale du programme est représentée par la figure 55 :

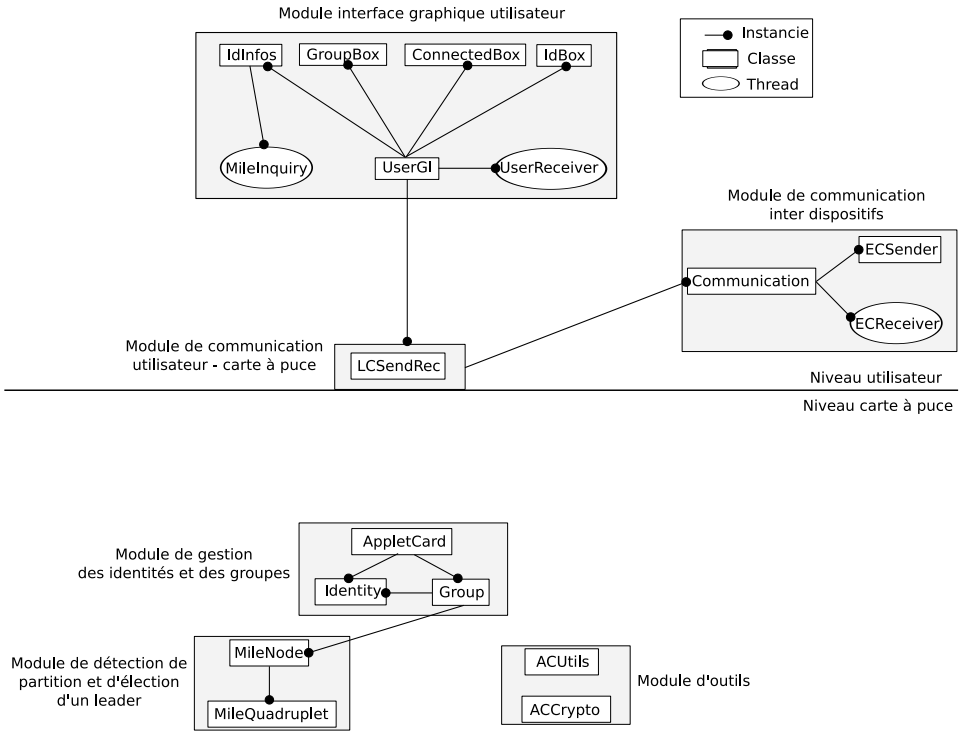


FIG. 55 – Squelette de l'implémentation SManet

4.2.3 Création d'une identité

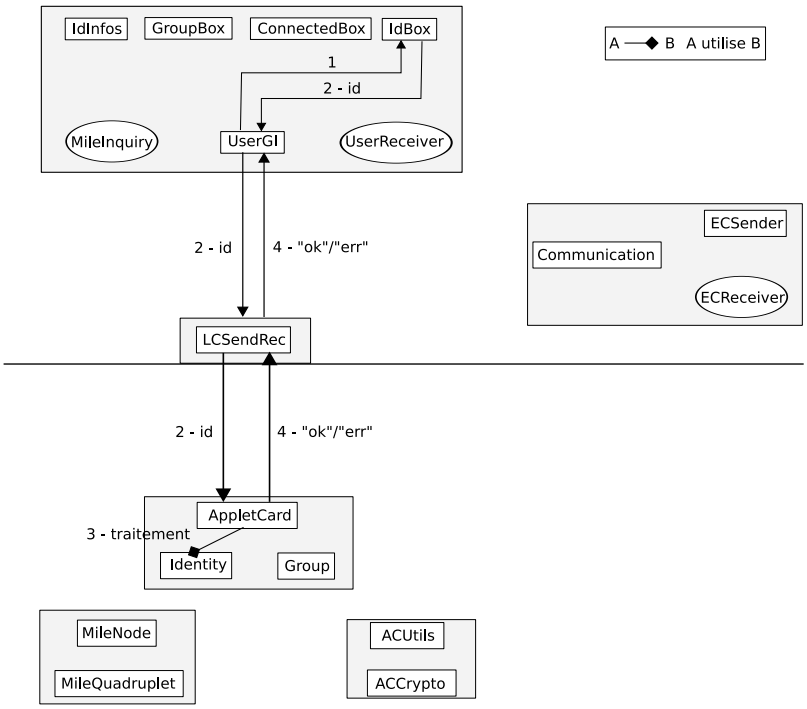


Fig. 56 – Implémentation de la création d'une identité

La création d'une identité passe par les étapes suivantes (fig. 56) :
- 1 - l'utilisateur ouvre la fenêtre de création d'une identité. Il doit y entrer son pseudonyme et valider ;
- 2 - le pseudonyme est envoyé à la carte à puce ;
- 3 - la carte vérifie que le pseudonyme proposé n'existe pas déjà. Si c'est le cas, elle prépare une réponse « err », sinon elle prépare un objet « Identity » en lui fournissant le pseudonyme reçu et une paire de clés qu'elle génère (cf. section 2.3.3 page 61), puis la réponse « ok » ;
- 4 - la réponse est envoyée à la partie utilisateur.

4.2.4 Création d'un groupe

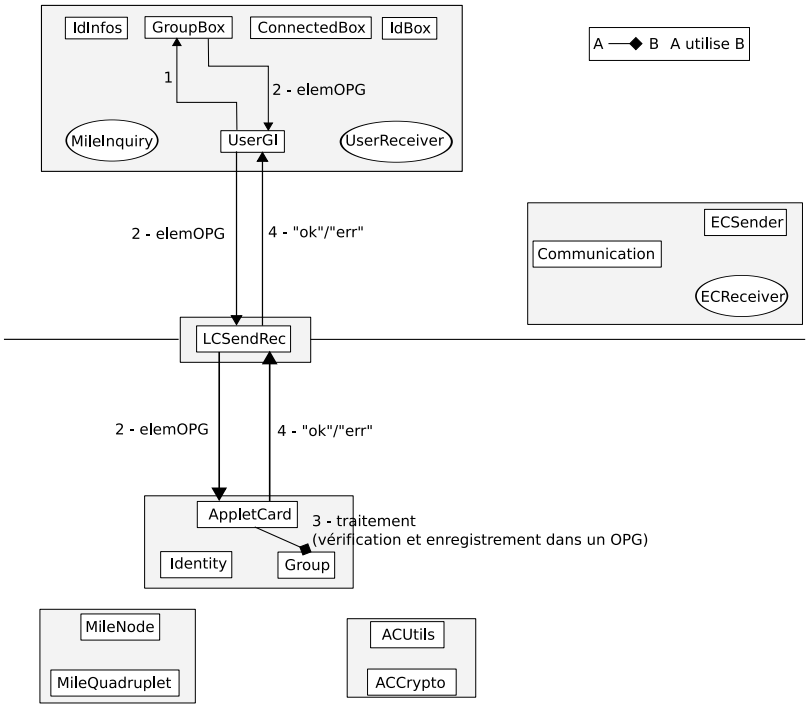


FIG. 57 – Création et enregistrement d'un groupe

La création d'un groupe passe par les étapes suivantes (fig. 57) :
- 1 - l'utilisateur ouvre la fenêtre de création d'un groupe. Il doit remplir un ensemble de champs ;
- 2 - les informations fournies par l'utilisateur sont envoyées à la carte (*elemOPG* sur la figure 57 : pseudonyme du groupe, description, option, niveau de restriction d'accès, [question *challenge*], [réponse au *challenge*], mode d'admission, [nombre de votes pour entrer], nombre maximum de plaintes acceptées, identité associée) ;
- 3 - la carte procède aux vérifications des données (si l'identité à associer existe, si les données ont le bon format, etc.). Si l'un des paramètres n'est pas valide, elle prépare la réponse « err », sinon elle enregistre les données dans un OPG et prépare la réponse « ok » ;
- 4 - la réponse est renvoyée à la partie utilisateur.

4.2.5 Découverte des groupes voisins (*Inquiry*)

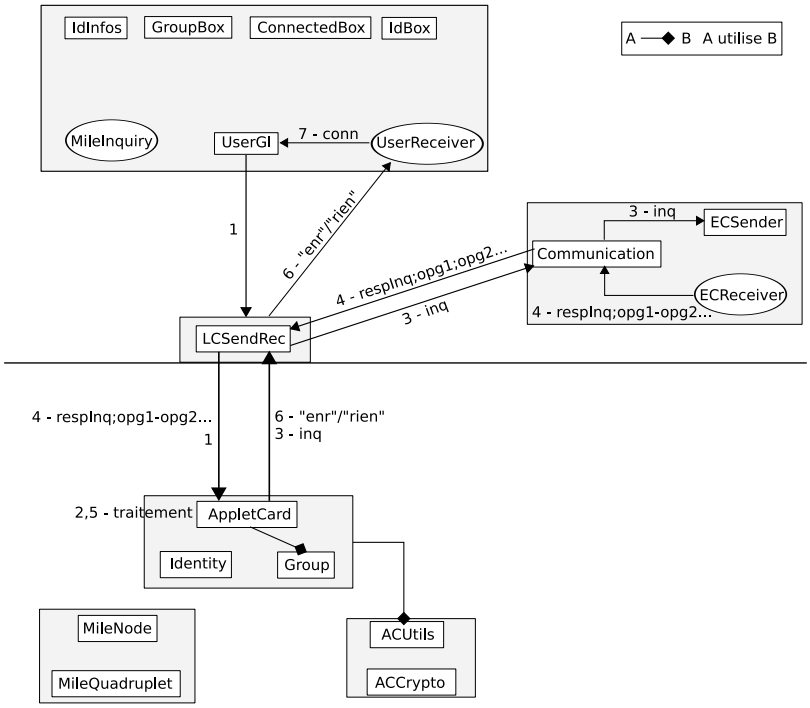


Fig. 58 – Architecture du processus de découverte des voisins (*inquiry*)

La découverte des voisins passe par les étapes suivantes (fig. 58) :
- 1 - l'utilisateur transmet à sa carte à puce une demande de découverte des voisins ;
- 2 - la carte prépare une requête à envoyer sur le canal inter cartes à puce (canal chiffré) ;
- 3 - cette requête (*inq*) est envoyée à l'interface carte/utilisateur, puis au module de communication qui la diffuse (technologie wifi 1.2.2 page 12) ;
- 4 - les cartes des voisins renvoient la liste des OPG des groupes dans lesquels ils sont actifs ;
- 5 - la carte d'origine enregistre les réponses dans des OPG (pour les groupes qui ne sont pas déjà enregistrés). Elle prépare la réponse « enr » si de nouveaux groupes ont été ajoutés ou « rien » sinon ;
- 6 - la réponse est transmise à la partie utilisateur.

4.2.6 Connexion à un groupe

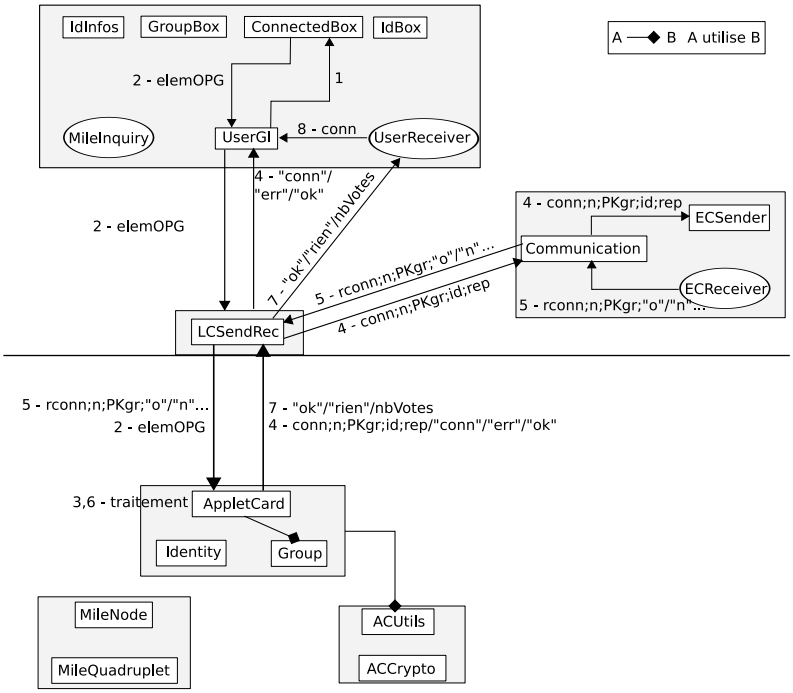


FIG. 59 – Implémentation d'une connexion à un groupe

La connexion à un groupe passe par les étapes suivantes (fig. 59) :

- 1 - l'utilisateur ouvre la fenêtre de connexion à un groupe et sélectionne l'identité de celui auquel il souhaite participer. Si l'accès à ce groupe est restreint, l'utilisateur entre sa réponse à la question d'entrée ;
- 2 - les informations fournies par l'utilisateur (*elemOPG*) sont envoyées à la carte ;
- 3 - la carte vérifie que le groupe existe (sinon elle prépare le message « err ») et qu'on n'y est pas déjà connecté (sinon elle prépare le message « conn »). Deux cas de figure peuvent se présenter :

  1. si le groupe est à accès ouvert :
     - 3 - l'utilisateur est immédiatement admis et la carte prépare une réponse « ok » ;
     - 4 - la réponse est renvoyée à la partie utilisateur.
  2. si le groupe est à accès restreint et que la carte possède le secret d'entrée :
     - 3 - la carte le compare à celui proposé par l'utilisateur. Si la réponse est correcte la carte prépare une réponse « ok », sinon elle prépare une réponse « err » ;
     - 4 - les réponses d'erreurs ou d'admission sont renvoyées à la parties utilisateur.
  3. si le groupe est à accès restreint et que l'admission doit être réalisée par vote des nœuds voisins :
     - 3 - la carte prépare une requête de demande de vote qui sera transmise par le canal sécurisé des cartes à puce ;

- 4 - la requête de connexion est envoyée à l'interface carte-utilisateur, puis au module de communication qui la diffuse ;
- 5 - les cartes des voisins envoient leur réponse ;
- 6 - la carte du candidat récupère les votes : s'il s'agit d'un vote négatif, elle prépare une réponse « rien » ; si le vote est positif, elle incrémente le nombre de votes déjà reçus et vérifie si le seuil nécessaire est atteint. Si c'est le cas, elle prépare une réponse « ok », sinon elle prépare une réponse avec le nombre de votes obtenus ;
- 7 - la réponse est envoyée à la partie utilisateur.

4.2.7 Activation d'un groupe

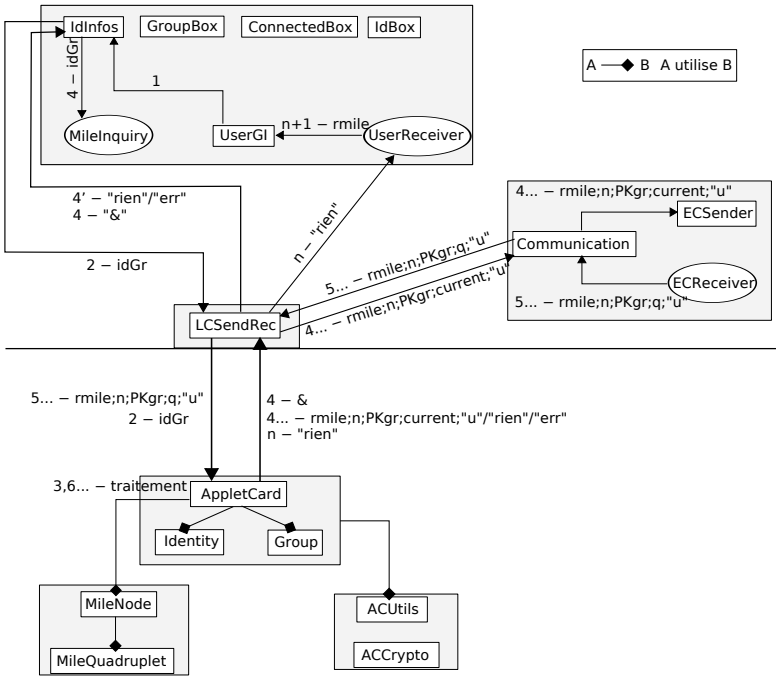


Fig. 60 – Implémentation de l'activation d'un groupe

L'activation de la participation à un groupe passe par les étapes suivantes (fig. 60) :

- 1 - l'utilisateur ouvre la fenêtre de participation aux groupes auxquels il est connecté, choisi un groupe auquel il souhaite participer et clique sur le bouton *Activate* correspondant ;
- 2 - la requête est envoyée à la carte avec l'identifiant du groupe sélectionné ;
- 3 - la carte vérifie que l'utilisateur est bien membre du groupe demandé, sinon elle prépare la réponse « err ». Si les conditions sont remplies, la carte passe le groupe à *actif*. Deux cas de figure sont ensuite possibles :
  1. si l'option du *leader* n'a pas été sélectionnée :
     - 3 - la carte prépare une réponse « rien » ;
     - 4 - la réponse est envoyée à la partie utilisateur ;

2. si l'option du *leader* a été sélectionnée :
   - 3 - la carte initialise un objet *MileNode* et prépare un message pour envoyer son quadruplet (cf. 3.3.3(a) page 95) aux membres voisins ;
   - 4 - la requête est envoyée à l'interface carte-utilisateur, puis au module de communication qui la diffuse.
   - 4' - en parallèle la carte renvoie une réponse spéciale à la partie utilisateur (le caractère &) afin qu'elle exécute le *thread MileInquiry* (en lui fournissant l'identifiant du groupe *idGr*). Celui-ci est chargé d'informer régulièrement la carte qu'elle doit envoyer une requête de découverte des quadruplets des membres voisins ;
   - 5 - les cartes des voisins envoient leurs quadruplets. La carte considérée exécute l'algorithme d'élection du *leader*. Si son propre quadruplet est modifié, la carte le diffuse de nouveau. Cette opération est répétée jusqu'à ce que le quadruplet ne soit plus modifié et la carte retourne alors « rien » ;
   - n - cette réponse (« rien ») est envoyée à une classe spéciale de la partie utilisateur qui est chargée d'orienter les messages générés par la carte suite à une requête d'une carte voisine ;
   - n+1 - la classe spéciale transmet l'information à la classe concernée.

Si l'option *leader* est sélectionnée, lors de l'activation d'un groupe nous avons vu que le *thread MileInquiry* était lancé. Ceci est décrit en figure 61.

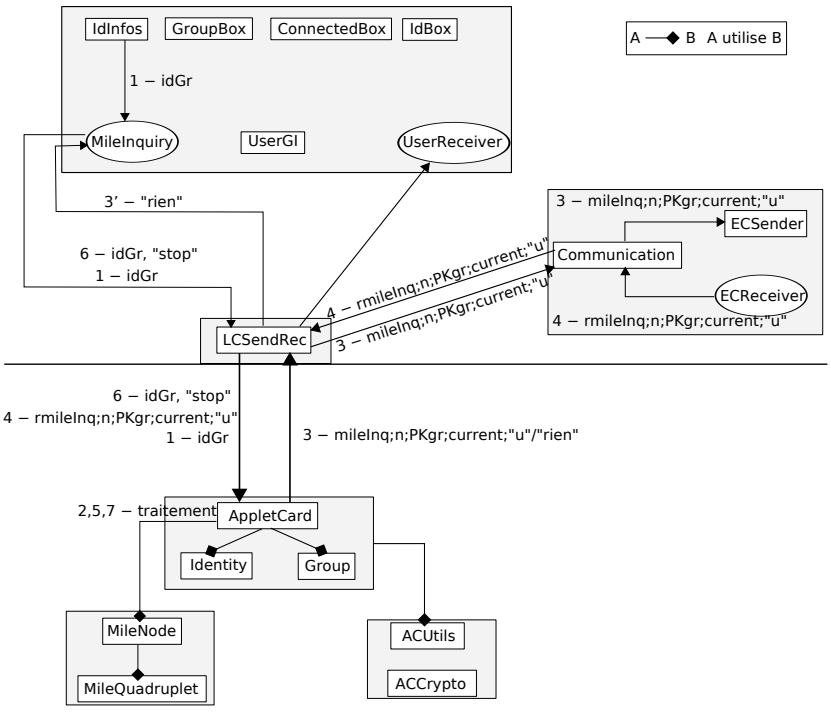


FIG. 61 – Implémentation de la découverte régulière des membres voisins dans le cadre de l'algorithme du *leader*

Les opérations alors réalisées sont les suivantes :

- 1 - le *thread* envoie à la carte une requête de découverte des quadruplets (cf. 3.3.3(a) page 95) voisins pour un groupe donné ;
- 2 - la carte procède aux vérifications nécessaires. Si tout va bien, elle prépare une requête de découverte des quadruplets des voisins membres du groupe ;
- 3 - la requête est envoyée à l'interface carte-utilisateur, puis au module de communication qui la diffuse ;
- 4 - les cartes des voisins répondent à la requête en renvoyant l'état de leur quadruplet ;
- 5 - La carte stocke les réponses qu'elle reçoit mais ne fait rien tant qu'elle ne reçoit pas l'ordre d'exécuter l'algorithme avec les réponses qu'elle a collecté. Ceci permet d'avoir une vision de l'ensemble des quadruplets voisins à un moment donné et de pouvoir appliquer les règles qui s'appuient sur la connaissance de tous les quadruplets voisins ;
- 6 - l'ordre arrive, au bout d'un laps de temps fixé par le développeur, du *thread* *MileInquiry* ;
- 7 - la carte exécute l'algorithme d'élection du *leader*. S'il y a un changement de son quadruplet courant, on procède aux même opérations que celles numérotées de 4 à n de l'activation d'un groupe. Il s'agit d'avertir les voisins de tout changement du quadruplet courant.

4.2.8 Découverte des voisins membres d'un groupe

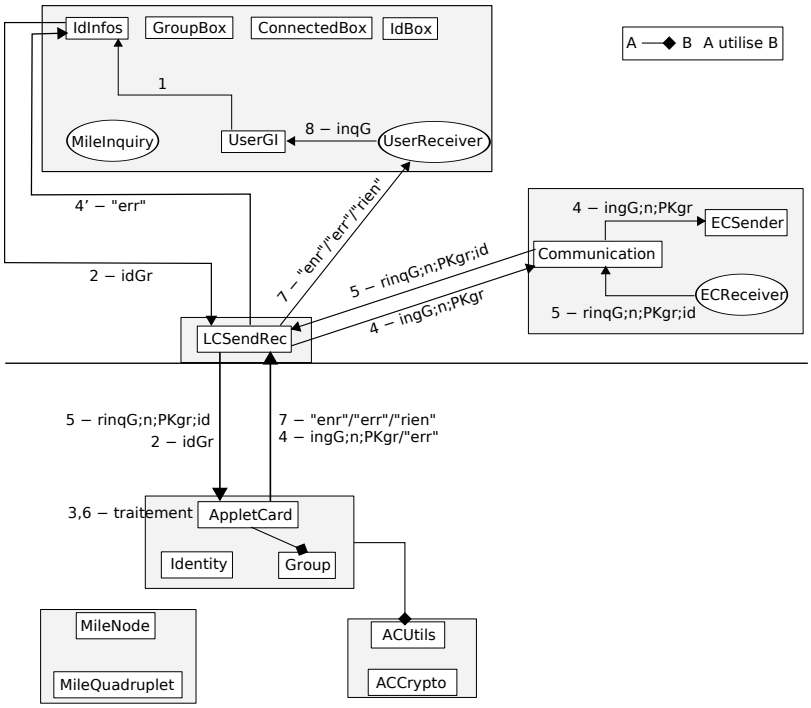


Fig. 62 – Recherche des voisins du même groupe

La découverte des voisins membres d'un groupe comporte les étapes suivantes (fig. 62) :
- 1 - l'utilisateur ouvre la fenêtre donnant les détails des groupes auxquels il est connecté ; il clique sur le bouton *Inquiry* correspondant au groupe qu'il souhaite interroger ;
- 2 - les informations sont envoyées à la carte à puce ;
- 3 - la carte vérifie que l'utilisateur est bien membre du groupe.
  1. si l'utilisateur n'est pas membre du groupe :
     - 3 - la carte prépare une réponse « err » ;
     - 4 - la réponse d'erreur est envoyée à la partie utilisateur.
  2. si l'utilisateur est membre du groupe :
     - 3 - elle prépare une requête à envoyer sur le canal inter cartes à puce (chiffrée) ;
     - 4 - la requête est envoyée à l'interface carte-utilisateur, puis au module de communication qui la diffuse ;
     - 5 - les cartes des voisins renvoient l'identité publique sous laquelle ils participent au groupe ;
     - 6 - la carte d'origine enregistre les réponses (sauf si elles sont déjà enregistrées). Elle prépare la réponse « enr » si de nouvelles identités ont été ajoutées ou la réponse « rien » sinon ;
     - 7 - la réponse est envoyée à la partie utilisateur ;

4.2.9 Echange d'un secret entre deux éléments d'un groupe

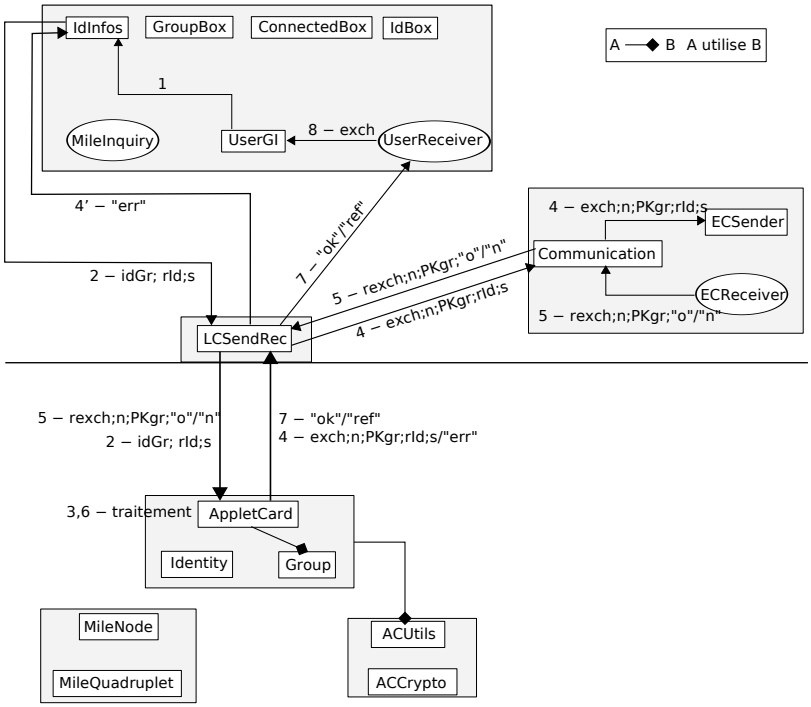


FIG. 63 – Echange d'un secret

Un élément veut envoyer un message à un autre qu'il a découvert dans son voisinage.

Nous décrivons le cas où il s'agit d'un secret à partager et qu'il faut conserver dans la carte, associé à l'identité des deux entités concernées (cf. 2.4.3(e), protocole traitant des échanges de données point à point page 79). S'il s'agit d'un simple message (qui ne doit pas être conservé) le processus est le même mis à part qu'il n'y aura pas d'enregistrement du message et le récepteur ne retournera aucune réponse.

Le processus mis en œuvre est décrit figure 63 :

- 1 - l'utilisateur ouvre la fenêtre donnant les détails des groupes auxquels il est connecté ; dans la liste des membres voisins (ils apparaissent après lancement d'un *inquiry*) il sélectionne celui avec lequel il souhaite communiquer, puis clique sur le bouton pour les communications privées ;
- 2 - les informations sont envoyées à la carte ;
- 3 - la carte vérifie que l'utilisateur est bien membre du groupe demandé.
  1. si l'utilisateur n'est pas membre du groupe :
     - 3 - la carte prépare la réponse « err » ;
     - 4 - la réponse d'erreur est envoyée à la partie utilisateur.
  2. si l'utilisateur est membre du groupe :
     - 3 - la carte prépare une requête pour l'échange du secret ;
     - 4 - la requête est envoyée à l'interface carte-utilisateur, puis au module de communication qui la diffuse.
     - 5 - la carte du voisin concerné fait passer la demande à son utilisateur puis transfère la réponse de ce dernier à la carte d'origine ;
     - 6 - la carte prépare une réponse pour son utilisateur, « ok » si la réponse du voisin est positive, « ref » sinon ;
     - 7 - cette réponse est envoyée à une classe spéciale de la partie utilisateur qui est chargée d'orienter les messages générés par la carte suite à une requête d'une carte voisine ;
     - 8 - la classe spéciale transmet l'information à la classe concernée.

4.2.10 Communication de groupe

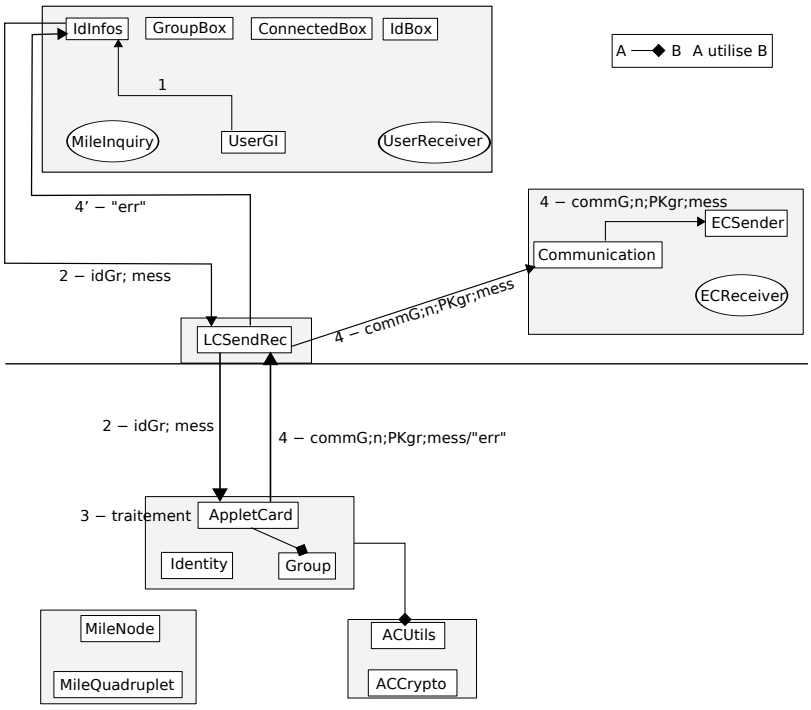


FIG. 64 – Communication de groupe

Quand un utilisateur veut envoyer un message à tout le groupe, le processus appliqué est celui décrit figure 64 :

- 1 - l'utilisateur ouvre la fenêtre donnant les détails des groupes auxquels il est connecté. Il clique ensuite sur le bouton pour demander une communication de groupe ;
- 2 - les informations sont envoyées à la carte ;
- 3 - la carte vérifie que l'utilisateur est bien membre du groupe demandé.
  1. si l'utilisateur n'est pas membre du groupe :
     - 3 - la carte prépare la réponse « err » ;
     - 4 - la réponse d'erreur est envoyée à la partie utilisateur.
  2. si l'utilisateur est membre du groupe :
     - 3 - la carte prépare une requête de communication de groupe ;
     - 4 - la requête est envoyée à l'interface carte-utilisateur, puis au module de communication qui la diffuse.

4.3 Implémentation au niveau utilisateur

Nous étudions maintenant la composition de chaque module que nous avons implémenté et expliquons une partie du code. Rappelons que nous développons en langage C# et que nous utilisons l'environnement *Visual Studio* et ses extensions pour les applications embarquées. Ces outils sont présentés en 4.1.1 et 4.1.2 page 110.

4.3.1 Interface utilisateur

La partie graphique de l'application est composée de cinq interfaces :
- l'interface principale où se trouve la liste des groupes voisins, la liste des identités créées, un bouton *inquiry* pour lancer la découverte des groupes voisins et des boutons d'accès aux autres interfaces (copie d'écran en 5.1.1 page 149).
- une interface de création des identités ;
- une interface de création des groupes ;
- une interface d'inscription à un groupe ;
- une interface pour la consultation des paramètres des groupes auxquels on est inscrit.

Pour le développement de ces interfaces nous avons utilisé les outils de développement graphique de *Visual Studio*. Le détail du fonctionnement de ces interfaces est présenté chapitre 6 « SManet : utilisation de l'application » dans la partie 5.1 (page 149) présentant le mode d'emploi de l'interface graphique.

4.3.2 Module de communication

Ce module sert à la communication entre deux dispositifs. Il est composé d'une classe principale *Communication.cs* pilotant une classe émetteur *ECSender.cs* et une classe récepteur *ECReceiver.cs*. Nous rappelons que le préfixe EC indique qu'il s'agit d'un module de communication avec des dispositifs extérieurs (*External Communications*). La technologie sans-fil que nous utilisons pour ces communications est la technologie Wi-Fi (présentée en 1.2.2 page 12).

Pour l'émetteur, il s'agit de créer une *socket* en utilisant les primitives de l'espace de noms System.Net.Sockets :

```
Socket sender = new Socket(AddressFamily.InterNetwork,
                           SocketType.DGram,
                           ProtocolType.Udp);

sender.SetSocketOption(SocketOptionLevel.Socket,
                       SocketOptionName.Broadcast,
                       1);
```

Le *broadcast* est réalisé de la manière suivante :

```
IEndPoint p = new IPEndPoint(IPAdress.Broadcast,  port);
sender.SendTo(buffer,  p);
```

buffer est un tableau de *bytes* et *p* représente une extrêmité du réseau, par exemple une adresse IP et un numéro de port.

Pour le récepteur, il faut aussi créer une *socket* :

```
Socket receiver = new Socket(AddressFamily.InterNetwork,
                             SocketType.DGram,
                             ProtocolType.Udp);

IPEndPoint p = new IPEndPoint(IPAddress.Any, port);
```

```
socket.Bind(p);
EndPoint ep = (EndPoint)p;
```

La réception se fait dans la méthode *Run* d'un *thread* par l'appel :

```
receiver.ReceiveFrom(buffer, ref ep);
```

Après la réception, les données sont envoyées directement à la carte à puce (le message est précédé d'un symbole lorsqu'il s'agit d'un message destiné à la carte à puce). Ceci est décrit ci-après.

4.3.3 Module d'interface carte/utilisateur

Ce module permet à l'utilisateur de communiquer avec la carte à puce en lui envoyant des données et en réceptionnant ses réponses.

Bibliothèque de communication carte - dispositif. Le standard de communication entre un dispositif (PC, PDA, etc) et une carte à puce est l'API PC/SC [121]. Dans notre cas, ces échanges sont réalisés grâce à l'API *SpringCard* [142] pour .NET fournie avec notre lecteur de cartes à puce et qui est une implémentation plus légère que PC/SC. Cette API a la particularité de fonctionner sous n'importe quel système. Elle est utilisable sous .NET en C# (cf. 4.1.2 page 110).

La classe *Apdu*. Cette classe fournit les primitives nécessaires pour créer l'interface de communication avec la carte locale et préparer (conversion en tableau de *bytes*) puis envoyer les données. Elle utilise les primitives de la bibliothèque *SpringCard*.

i) La fonction *ReaderOpen* récupère le nom du lecteur de carte avec lequel on va communiquer :

```
SCARD.EstablishContext(SCARD.SCARD_SCOPE_SYSTEM, null, null,
                       ref SCardContext);

SCARD.ListReaders(SCardContext, null, SCardReaders, 1024);

return SCARD.MultiStringItem(SCardContext, SCarReaders, 0);
```

La primitive *EstablishContext* établit le contexte dans lequel devront se dérouler les communications (avec des applications utilisateurs ou des applications systèmes). Dans cette bibliothèque les trois premiers paramètres ne sont pas utilisés et restent toujours les mêmes et le quatrième est une référence à ce contexte.

La primitive *ListReaders* permet de retourner la liste de tous les lecteurs disponibles. Elle prend en premier paramètre le contexte établi précédemment. Le deuxième paramètre n'est pas utilisé. Le troisième paramètre est un tableau de caractères à allouer par le développeur et qui sera rempli par le nom des lecteurs disponibles. Enfin, le quatrième paramètre est la taille de ce tableau.

La primitive *SCardMultiStringItem* retourne une chaîne de caractères provenant d'un tableau de chaînes. Le premier paramètre est le contexte établi au début ; le deuxième paramètre est le tableau de chaînes et le troisième est simplement l'index dans le tableau de la chaîne de caractères à retourner. En l'occurrence, nous souhaitons obtenir la première entrée (nous n'utiliserons ici qu'un seul lecteur de carte).

ii) Une deuxième fonction *ReaderClose* utilise simplement la primitive SCARD.ReleaseContext qui prend en paramètre la référence au contexte et permet la destruction de celui-ci.

iii) La troisième fonction *CardConnect* prépare et réalise la connexion avec le lecteur de carte.

```
SCARD.Connect(SCardContext, readerName, SCARD.SCARD_SHARE_EXCLUSIVE,
              SCARD.SCARD_PROTOCOL_Tx, ref SCardHandle,
              ref SCardProtocol)
```

Explication des paramètres :

- le premier paramètre est le contexte établi auparavant ;
- le deuxième est le nom du lecteur converti en tableau de caractères ;
- le troisième paramètre peut être SCARD_SHARE_SHARED (on autorise l'utilisation de la carte par d'autres applications) ou SCARD_SHARE_DIRECT (le lecteur concerné est dédié aux communications avec l'application utilisant ce paramètre) ; dans cette API il n'est pas utilisé et est toujours positionné à SCARD_SHARE_EXCLUSIVE (on n'autorise pas l'utilisation de la carte par d'autres applications) ;
- le quatrième est le protocole de transmission choisi : SCARD_PROTOCOL_T0 pour un protocole orienté octets, SCARD_PROTOCOL_T1 pour un protocole orienté paquets, PROTOCOL_Tx pour octet ou paquet. Ces drapeaux peuvent ensuite être combinés avec trois autres drapeaux : SCARD_PROTOCOL_PTS (pour forcer la vitesse à 9600 bauds au lieu de la plus haute disponible), SCARD_PROTOCOL_HIS (pour doubler la vitesse d'horloge) ou SCARD_PROTOCOL_EMV (qui force les cartes bancaires utilisant les protocoles EMV à se conformer aux protocoles orientés octets ou paquets). Il existe aussi un drapeau SCARD_PROTOCOL_RAW pour les cartes synchrones (cartes à mémoire uniquement, dites synchrones car le protocole de dialogue est tel que toute requête est traitée immédiatement).

On utilise ensuite la primitive *SCARD.Status* pour récupérer l'ATR (*Answer To Reset*, code de réponse de la carte suite à une ré-initialisation) de la carte de la manière suivante :

```
SCARD.Status(SCardHandle, null, ref i, ref cardState, ref
             SCardProtocol, atr, ref SCARD.SCARD_ATR_LENGTH);
```

iv) Une quatrième fonction permet l'échange de données entre le terminal et la carte. Elle prépare l'APDU réponse en remplissant le tableau de *bytes* et en faisant pour cela les conversions nécessaires, puis elle utilise la primitive suivante :

```
SCARD.Transmit(SCardHandle, ADPU, APDULen, buffer, 500):
```

Explication des paramètres :

- le premier paramètre est le *handle* (identifiant unique) de la connexion établie dans SCardConnect ;
- le deuxième est le tableau d'APDU ;
- le troisième est la longueur du tableau d'APDU ;
- le quatrième est le tableau dans lequel sera stockée la réponse APDU de la carte ;
- enfin le cinquième est la longueur de la réponse.

v) La dernière fonction permet de terminer la connexion entre la carte et le terminal par l'appel de la primitive :

```
SCARD.Disconnect(SCardHandle, SCARD.SCARD_UNPOWER_CARD);
```

Explication des paramètres :

- le premier paramètre est le *handle* de la connexion établie dans SCardConnect ;
- le deuxième permet d'arrêter la carte mais pas le lecteur. Le drapeau SCARD_LEAVE_CARD ferme uniquement le *handle*, la carte restant dans le même état ; SCARD_RESET_CARD remet la carte à zéro mais ne coupe pas son alimentation alors que SCARD_EJECT_CARD coupe l'alimentation de la carte (si les deux slots ne sont plus alimentés, alors le lecteur non plus ; il sera rallumé au prochain *SCardConnect*).

La classe *LCSendRec.* Cette classe se charge de transmettre les messages entre le terminal et la carte et les messages inter cartes via le module de communication avec l'extérieur. Le préfixe *LC* indique qu'il s'agit d'un module de communications locales (*Local Communications*).

La fonction *sendApplet(string message, string ins)* fait appel à la suite de fonctions *ReaderOpen*, *CardConnect* et *ExchangeAPDU* pour instancier le module de communication vers la carte locale. Ensuite elle récupère la réponse provenant de la carte dans un tableau de *byte* et la convertit en chaîne de caractères. Enfin, selon les en-têtes de la réponse, elle envoie les données récupérées à traiter soit à la partie utilisateur, soit au module de communication vers l'extérieur pour qu'il les diffuse. Cette fonction est rendue atomique par l'emploi d'un *mutex* afin que les réponses aux requêtes soient traitées en totalité avant que toute autre requête à la carte soit effectuée.

```
// Méthode d'envoi des messages à la carte à puce
public string sendApplet(string message, ins)
{
  mutex.WaitOne();
  byte response = apdu.ExchangeAPDU(message, ins);
  string strResponse = ASCIIEncoding.GetString(response, 0,
                                                response.Length - 1);
  // traitement des requêtes
  mutex.ReleaseMutex();
}
```

4.4 Implémentation au niveau carte à puce

Pour ce niveau, nous développons en langage *JavaCard* (cf. 4.1.4 page 112) sous la plateforme *eclipse* munie du *plug-in* JCOP (cf. 4.1.3 page 111).

4.4.1 Applet principale

Cette *applet* représente l'administrateur chargé de toutes les vérifications, validations et certifications de notre solution.

La classe *AppletCard* est la classe principale contenant tous les éléments nécessaires à la réception des données envoyées par le niveau humain. Lorsqu'une commande APDU arrive, le JCRE appelle la méthode *process*. Le squelette de cette méthode est le suivant :

```
// Déclaration des variables globales
...
byte[] response = null;
byte[] buf = apdu.getBuffer();
byte[] buftmp = new byte[dataSize];
...

// Fonction de réception des APDU sur la carte
...
byte lc = (byte)(buf[ISO7816.OFFSET_LC]);
byte dataSize = (byte)(apdu.setIncomingAndReceive());

if ((lc != -1 || dataSize != 1))
{
  ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);
}
Util.arrayCopy(buf, (short)ISO7816.OFFSET_CDATA), buftmp,
                (short)indexRcv, (short)buf[ISO7816.OFFSET_LC]);

switch(buf[ISO7816.OFFSET_INS])
{
  case (byte)0x..
  ...
  default:
  ISOException.throwIt (ISO7816.SW_INS_NOT_SUPPORTED);
  ...
}
...
```

La taille maximum des données à envoyer est limitée par la taille annoncée dans la variable *lc* qui est un hexadécimal sur 1 *byte* maximum (donc au plus 255). Dans notre cas, la taille des données à envoyer pouvant excéder la contenance maximale du *buffer apdu*, nous avons mis en place une procédure permettant de savoir si l'ensemble des données nécessaires à un traitement était arrivé. Si ce n'est pas le cas, les données déjà arrivées sont stockées temporairement dans un tableau de *bytes* qui se remplit au fur et à mesure. Une fois que le marqueur de fin de message apparaît la carte procède au traitement :

```
// Envoi d'une réponse APDU

// Copie du buffer APDU dans buftmp depuis l'indice indexRcv
Utils.arrayCopy(buf, (short)ISO7816.OFFSET_DATA, buftmp,
                        (short) indexRcv, (short)buf[ISO7816_LC];

if (buf[254] == CharCode.and)
{
  indexRcv += 254;
  // réponse à renvoyer signifiant que la carte attend la suite
  //des données
  response[0] = CharCode.and;
}
else // exécution normale du programme
{
  ...
}
```

De la même manière si la taille des données à envoyer de la carte au terminal est trop importante, il faut découper la réponse en plusieurs paquets à envoyer en plusieurs fois :

```
// Déclaration des variables globales
byte[] tmp1;
...

// Constructeur
tmp1 = new byte[255];

// Méthode process, envoi de la réponse
...
short le = apdu.setOutgoing();
short lengthToSend = 0;
short stillToSend = response.length - 254*isToSend; // reste à envoyer
short indexSend = 254*isToSend;                     // index de début
boolean isTooLong = false;

if (le < response.length || stillToSend > 254)
{
  tmp1[0] = (byte)CharCode.and;
  Util.arrayCopy(tmp1, (short)0, buf, (short)0, (short)1);
  isTooLong = true;
}

// si le message est trop long envoi du nième bloc de 255 bytes
if (isTooLong)
{
  Util.arrayCopy(response, (short)indexSend, buf, (short)0,
                 (short)254);
  lengthToSend = 255;
  isToSend++;
}
// sinon on envoie tout le reste
else
{
  Util.arrayCopy(response, (short)indexSend, buf, (short)0,
```

```
                    (short)stillToSend);
    lengthToSend = stillToSend
  }
  apdu.setOutgoingLength((short)lengthToSend);
  apdu.sendBytes((short)0, (short)lengthToSend);
  ...
```

CharCode.and est la représentation dans la carte à puce du symbole '&' (cf. 4.4.3(b) page 134).

4.4.2 Algorithme d'élection du *leader*

Rappelons tout d'abord que notre algorithme d'élection du *leader* repose sur la connaissance par chaque nœud d'un plus court chemin jusqu'au *leader* (déterminé par un nombre de sauts et par l'identité du nœud précédent dans ce chemin si on part du *leader* vers les nœuds les plus éloignés - de la racine de l'arbre du plus court chemin vers les feuilles). L'état d'un nœud est défini par un quadruplet composé de l'identité du *leader*, de l'identité du nœud lui-même, du nombre de sauts jusqu'au *leader*, du prédécesseur ainsi que de la date à laquelle le message a été envoyé.

Le module d'élection du *leader* est composé de quatre classes préfixée par le mot *Mile* pour *Mobility adapted Implementation of Leader Election* :

- *MileQuadruplet* est la classe qui définit les quadruplets, c'est-à-dire l'état d'un nœud à un moment donné ;
- *MileCriteriaElection* implémente le critère d'élection, c'est-à-dire qu'elle se charge d'attribuer la valeur de leur critère à chaque nœud (puissance de calcul, énergie consommée, etc.) Elle se charge aussi des comparaisons entre les valeurs ;
- *MileNode* est la classe principale ;
- *MileInquiry* est une classe se trouvant au niveau utilisateur. Elle implémente un *thread* (les cartes que nous utilisons ne permettent pas de le faire elles-mêmes) chargé de demander à la carte d'envoyer une découverte des quadruplets des voisins régulièrement (intervalle de temps fixé).

Remarque : comme indiqué en 3.2.3 (paragraphe « Problématique pour la découverte des voisins » page 92) nous avons mis en place un système de vérification pour que les requêtes envoyées par *MileInquiry* à la carte ne soient pas détournées. Lorsque l'*applet* principale reçoit un certain nombre de fois le quadruplet d'un même voisin, elle vérifie qu'elle a elle-même lancé une découverte du voisinage. Si ce n'est pas le cas cela signifie que, alors qu'une carte voisine a lancé plusieurs fois une requête de découverte des voisins, la carte considérée elle n'en a pas lancé. Elle n'a donc pas reçu les requêtes de la part du *thread MileInquiry*. Elle considère alors que la partie utilisateur triche et elle invalide l'accès au groupe.

4.4.3 Module outils

a) Outils cryptographiques.

Ce module fournit les outils nécessaires aux méthodes cryptographiques. On y retrouve donc une fonction pour la génération des clés :

```
KeyPair keyPair = new KeyPair(KeyPair.ALG_RSA,
                              KeyBuilder.LENGTH_RSA_1024);
keyPair.genKeyPair();
publicKey = (RSAPublicKey)keyPair.getPublic();
privateKey = (RSAPrivateKey) keyPair.getPrivate();
```

Nous avons aussi besoin d'une fonction pour la signature des messages :

```
// Déclaration des variables globales
...
Signature sig;
byte[] signedData;
...

// Constructeur
...
sig = Signature.getInstance(Signature.ALG_RSA_SHA_PKCS1, false);
signedData = new byte[MAX_MESSAGE];
...

// Méthode statique de signature
...
sig.init(RSAPrivateKey, Signature.MODE_SIGN);
short sizeSig = sig.sign(dataToSign, (short)0, (short)(lc),
                                                signedData, (short)0);
```

b) Outils pour la gestion des caractères.

Le type caractère n'existant pas, nous avons dû faire une table de correspondance entre les valeurs hexadécimales et un certain nombre de caractères alphabétiques dans la classe appelée *CharCode*.
Nous avons aussi créé une classe *MileUtils* dans laquelle nous avons implémenté un ensemble de fonctions statiques pour manipuler des tableaux à la manière d'une chaîne de caractères, i.e. avec un élément de fin (concaténation, découpage d'une chaîne en plusieurs sous chaînes en fonction d'un symbole, etc).

4.5 Tests et évaluations

Nous avons testé notre code en utilisant deux approches. Dans la première les cartes à puce sont émulées. Le code est néanmoins exactement le même que celui qui serait réellement installé sur une carte à puce (excepté les parties spécifiques aux échanges d'APDU). Cette version facilite l'observation des échanges entre cartes (sinon nous n'avons pas accès aux données contenues dans celles-ci).
Dans la seconde approche nous utilisons réellement les cartes à puce. Nous verrons plus loin que ces tests ont été très limités.

Les tests ont été effectués sur des PDA *Dell Axim x51v* avec communications WiFi. Pour les tests avec cartes à puce nous avons utilisé des lecteurs *Pro-Active SpringCard-CF*

	Nombres de groupes possibles	
	1 groupes	**10 groupes**
Module principal (total de l'espace réservé)	**19,05 ko**	**962,84 ko**
Module identité	2,05 ko	2,05 ko
Module groupe	6,09 ko	44,26 ko
Module *leader*	0,12 ko	0,12 ko
Autres (crypto, outils, etc.	1,25 ko	1,25 ko
Taille du code	**20 ko**	
Espace total nécessaire	**$\simeq$ 39 ko**	**$\simeq$ 983 ko**

Tab. 32 – Tableau récapitulatif de la mémoire utilisée par le module administrateur

et des cartes *Philips JCOP 20* (modèle P8WE5033 : 96ko de ROM, 32ko d'EEPROM, de 2ko de RAM, coprocesseurs Triple DES et FrameX RSA).

4.5.1 Utilisation mémoire

Les cartes à puce ayant des ressources très limitées, il est important de faire le point sur la place mémoire que peut prendre ce programme. Rappelons qu'à l'installation de l'*applet*, il est fortement recommandé d'allouer toute la mémoire nécessaire pour les objets de l'application dans la méthode *install* (ou dans le constructeur qu'elle appelle) à cause de l'absence de mécanisme de ramasse-miettes (cf. 4.1.4 page 112).

Nous avons listé l'ensemble des objets créés dans chaque classe et déterminé la place mémoire qu'ils utilisent. Ceci ajouté à l'espace mémoire occupé par le code en lui-même, nous avons ainsi pu calculer l'espace mémoire que nécessite la partie carte à puce de l'application. Les résultats sont présentés sur le tableau 32 ci dessus.

Nous voyons (tab. 32) que notre application nécessiterait plus de mémoire que n'en possèdent les cartes que nous utilisons (24ko libres pour les applications).
La capacité mémoire des cartes à puce est appelée à augmenter très rapidement, certaines ayant déjà 256ko d'EEPROM. Cependant, si les applications pour ces dispositifs évoluent elles aussi et demandent de plus en plus de mémoire, les problèmes dans ce type de dispositifs demeureront. Dans [43] une solution est présentée proposant de stocker des données sur le disque dur du terminal et non plus sur la carte. La solution gère un ensemble de clés cryptographiques assurant la confidentialité et l'intégrité des données.

4.5.2 Observation et validation des protocoles

Pour la validation de nos protocoles nous nous sommes servi de deux outils que nous présentons par la suite : des fichiers *logs* et le logiciel AVISPA. Nous les appliquons à chaque protocole et en expliquons les résultats.
Remarque : les fichiers AVISPA complets sont donnés en annexe C (page 205).

a) Fichiers *logs*

Dans la solution émulée, nous avons mis en place un système de création de fichiers *log* afin de contrôler si les données qui transitent sont les bonnes, si tous les messages sont bien envoyés, etc. Une ligne de *log* est composée du type de requête et des paramètres contenus dans le message.

Chaque requête et chaque réponse est identifiée par une lettre :

- a : requête de découverte des groupes voisins ;
- b : réponse à une requête de découverte des groupes voisins ;
- c : requête de découverte des identités des voisins d'un même groupe ;
- d : réponse à une requête de découverte des identités des voisins d'un même groupe ;
- e : requête de connexion à un groupe par vote des membres ;
- f : réponse à une requête de connexion à un groupe par vote des membres ;
- g : envoie du quadruplet courant de l'émetteur du message après sa mise à jour ;
- h : requête de découverte des quadruplets voisins (algorithme du *leader*) ;
- i : réponse à une requête de découverte des quadruplets des voisins (algorithme du *leader*) ;
- j : requête d'échange d'un secret avec un autre membre d'un groupe ;
- k : réponse à une requête d'échange d'un secret avec un autre membre d'un groupe ;
- l : requête d'envoi d'un message de groupe ;
- m : requête d'envoi d'une demande de plainte contre un nœud.

Nous décrivons maintenant les paramètres éventuellement utilisés :

- *seq_num* est un *nonce* utilisé pour identifier les échanges (pour vérifier qu'une réponse correspond à la bonne requête) ;
- *group_id* est le pseudonyme du groupe choisi par son créateur ;
- *group_key* est l'autre partie de l'identifiant (partie publique de la clé associée au pseudonyme) ;
- *owner_id* est le pseudonyme de l'émetteur du message ;
- *owner_key* est l'autre partie de l'identifiant du noeud émetteur (partie publique de la clé associée au pseudonyme) ;
- *ownerId* est l'identité publique complète de l'émetteur du message ;
- *dst_id* est le pseudonyme du nœud ciblé par le message ;
- *dst_key* est l'autre partie de l'identifiant du nœud ciblé par le message (partie publique de la clé associée au pseudonyme) ;
- *dstId* est l'identité publique complète du nœud ciblé par le message ;
- *type_conn* est le type de restriction et vaut *ouvert* ou *secret* ;
- *challenge* est la question à laquelle il faut répondre correctement pour être admis dans le groupe (0 si le groupe est ouvert) ;
- *response* est la réponse à la question précédente (0 si le groupe est ouvert) ;
- *message* est une chaîne de caractères à envoyer ;
- *type_lead* vaut *lead* si le créateur du groupe concerné a choisi de faire tourner l'algorithme du *leader* dans son groupe.
- *description* est la description du groupe qui permettra d'en connaître le but ;

- *type_adm* est le type d'admission choisi par le créateur, *a* pour une admission automatique par la carte par simple comparaison avec la réponse qui lui a été fournie, *b* pour une admission par vote des autres membres du groupe concerné ;
- *nb_to_adm* est le nombre de votes qu'il faut pour être admis dans le cas *b* ;
- *nb_to_excl* est le nombre de plaintes qu'il faut recevoir pour être exclu du groupe ;
- *vote* est la réponse à une demande de connexion à un groupe ('o' ou 'n') ;
- *quad* est un quadruplet dans le cadre de l'élection du *leader* (cf. 3.2 page 87) ;
- *state_mile* est le type de traitement d'un quadruplet reçu par l'algorithme, 'u' pour « update » et 'd' pour « delete » ;
- *clock* est l'«heure» de l'ilôt courant ;
- *v_leader* est la valeur du *leader* de l'îlot en cours.

b) Brève introduction à AVISPA

L'outil AVISPA [11] permet une analyse et une validation des protocoles de sécurité. Il détecte les failles de sécurité dans les protocoles modélisés grâce au langage formel HLPSL (*High-Level Protocol Specification Language*). On y spécifie les identifiants, les échanges du protocole, les connaissances globales et locales, la sécurité recherchée (comme le secret d'un message ou l'authentification d'une clé), etc.

Les bases de HLPSL. Un fichier utilisant le langage HLPSL est composé de plusieurs sections :

1. les sections *basics role* : elles représentent les participants au protocole, leurs connaissances, leurs communications, etc. Le squelette de ces sections est le suivant :

```
role membre (paramètres)
played_by id_membre def=

local
    val1_t1,val2_t1,..,valn_t1: type1,
    val1_t2,val2_t2,...,valn_t2: type2,
    ...
    val1_tn,val2_tn,...,valn_tn':typen

const
    const1_t1,const2_t1,..,constn_t1: type1,
    const1_t2,const2_t2,...,constn_t2: type2,
    ...
    val1_tn,val2_tn,...,valn_tn':typen

init
    State:=val_state

transitions
    1. State=val_state /\ RCV(start) =|>
                       ...
                       /\...
```

```
      State':=val_state' ...
                         /\...

   2. State=val_state /\...
                      /\...
      State':=val_state' ...
                         /\...
```

`end role`

Les paramètres sont les éléments publics accessibles par tous, comme par exemple les clés publiques en jeux dans le chiffrement des messages échangés, les identités des protagonistes de l'échange, les identifiants des canaux de communication (RCV et SND), etc. Les variables locales sont les éléments qui transitent entre deux nœuds et qui ne sont pas publics.

Il existe différents types de variables :
- $public\_key$: clé publique ;
- $symmetric\_key$: clé symétrique ;
- $hash\_func$: fonction de hachage ;
- $text$: chaine de bits non interprétée ;
- $message$: type général des messages ;
- set : collection de valeurs non ordonnées.

Les transitions contiennent des expressions pour exprimer le déroulement des échanges :
- $A := B$: affectation, A prend la valeur B ;
- $A = B$: égalité entre A et B ;
- $RCV(M)$ et $SND(M)$: réception et envoi d'un message ;
- M_K : chiffrement de M par la clé publique K ;
- $M\_inv(K)$: chiffrement de M par la clé privée associée à la clé publique ;
- var' : prochaine valeur de la variable dans la transition.

Les transitions contiennent aussi des fonctions pour tester la sécurité :
- $secret(var, id\_trans, m1, m2)$: spécifie que la variable var doit rester secrète entre les membres $m1$ et $m2$ sur la transition id_trans ;
- $witness(m1, m2, var, val)$ et $request(m2, m1, var, val)$: ces fonctions permettent d'authentifier l'origine des messages. La fonction $witness$ indique que m1 veut transmettre à $m2$ la valeur val dans la variable var. De l'autre côté, la fonction $request$ exprime le fait que $m2$ a reçu de $m1$ la valeur val dans la variable var et qu'il doit vérifier que le message provient bien de $m1$. Ces deux fonctions sont donc utilisées pour tester les attaques de type *Man-in-the-middle*.

2. les sections *Composed role* : elles représentent des scénarios utilisant les rôles basiques. Leur squelette est le suivant :

`role nom_role(paramètres) def=`

```
[définition des variables]

composition
    role1(paramètres) /\ role2(paramètres)
    /\ role3(paramètres)
    /\ ...
    /\ rolen(paramètres)

end role
```

En général, il y deux sections de rôles composés :
- *role session* : on y définit les rôles basiques qui entreront en jeu dans le protocole ;
- *role envionnement*, on défini l'environnement, à savoir :
  - les variables publiques ;
  - les connaissances de l'attaquant :

```
intruder_knowledge = {liste des variables}
```

  - la composition de sessions afin de décrire le protocole.

3. section *goal* : c'est dans cette section que l'on définit quels tests de sécurité on souhaite réaliser.
   - authentification de l'émetteur d'un message en fonction d'un élément *var* : elle est réalisée en utilisant, lors de l'expression des échanges, la fonction *witness* du côté de l'émetteur et la fonction *request* du côté du récepteur (fonctions présentées plus haut). Dans la section *goal* il faut alors ajouter la ligne :
     `authentication_on var`
   - secret d'un message *var* : elle est réalisée en utilisant la fonction *secret* (décrite plus haut) lors de l'échange de la variable du côté de l'émetteur. Dans la section *goal* on ajoute alors la ligne :
     `secrecy_of var`

Les spécifications des protocoles décrites dans ce langage sont ensuite traduites dans un format intermédiaire (FI), puis analysées selon quatre méthodes différentes : *On-the-fly Model-Checker* (OFMC), *Constraint-Logic-based Attack Searcher* (CL-AtSe), *SAT-based Model-Checker* (SATMC) et *Tree Automate based on Automatic Approximations for the Analysis of Security Protocols* (TA4SP). Les explications de ces méthodes sont données en [11]. Le résultat de ces analyses est donné dans un fichier (de format fonction du modèle employé) comportant un certain nombre de champs dont les principaux sont :
- $SUMMARIZE$: indique si le protocole est « safe » ou « unsafe » ;
- $GOAL$: rappel de ce qu'on avait demandé a protéger en HLPSL ;
- $BACKEND$: indique le modèle de vérification utilisé (OFMC, CL-AtSe, SATMC ou TA4SP).
- $ATTACK\ TRACE$ (si une attaque a été détectée) : donne le procédé de l'attaque, i.e. les différents échanges mettant en jeu l'attaquant et permettant cette attaque.

Pour des besoins de lisibilité, nous avons choisi de fournir le code AVISPA en annexe C (page 205). Il nous est apparu peu utile de donner des extraits de code dans cette section mais il est conseillé de la lire en parallèle de l'annexe C.

c) Validation des échanges entre cartes à puce

Echange de type 1. Dans ce type d'échange, un nœud A envoie une requête composée de l'identifiant de requête *Req* et d'un *nonce* N. Le récepteur B la traite et renvoie une réponse composée de l'idenfiant de réponse *Resp* correspondant, du *nonce* N et d'un message M qui doit rester confidentiel. Le premier échange ne contient pas de données critiques donc il est simplement signé. Dans la réponse le message doit rester confidentiel donc il est en plus chiffré.

```
1. A -> B : {PKa}SIGNca {Req, N}SIGNa
2. B -> A : {PKb}SIGNca {{Resp, N, M}PKa}SIGNb
```

Nous décrivons ce protocole dans un fichier AVISPA (donné en annexe C) qui nous permet de vérifier que tout message M est sûr c'est-à-dire qu'il reste confidentiel. Nous souhaitons aussi une authentification forte de la réponse en fonction du numéro de séquence n : on veut s'assurer que si la réponse reçue contient le bon numéro de séquence c'est qu'elle provient d'un nœud (valide) qui a reçu la requête.

Le résultat est le suivant :

```
SUMMARY
  SAFE
DETAILS
  BOUNDED_NUMBER_OF_SESSIONS
PROTOCOL
  /root/results/avispa_exch1.if
GOAL
  as_specified
BACKEND
  OFMC
COMMENTS
STATISTICS
  parseTime: 0.00s
  searchTime: 0.03s
  visitedNodes: 76 nodes
  depth: 6 plies
```

Ce fichier nous indique que pour les buts *as specified* pour $secrecy\_of\_m$ (i.e. l'élément m doit rester secret au cours de l'échange concerné) et $authentication\_on\_alice\_bob\_n$ (authentification de *Bob* auprès de *Alice* par l'élément n), le protocole explicité dans le fichier $avispa\_exch1.if$ est $SAFE$ par le modèle OFMC, c'est-à-dire qu'aucune faille de sécurité n'a été détectée par ce modèle (vérification itérative de chaque étape du protocole). Les résultats sont les mêmes pour les trois autres modèles.

Echange de type 2 Dans ce type d'échange, un nœud A envoie à un nœud B un message contenant des données critiques devant rester confidentielles. La requête contient donc ces

données ($M1$), l'identifiant de requête *Req* et un *nonce* N. Le message est reçu par B qui le traite et renvoie une réponse contenant elle aussi des données critiques. Cette réponse est donc composée de l'identifiant de réponse *Resp*, du *nonce* N reçu précédemment et des données ($M2$) à envoyer. Les deux échanges contiennent des données critiques donc ils doivent être signés et chiffrés.

```
1. A -> B : {PKa}SIGNca {{Req, N, M1}PKb}SIGNa
2. B -> A : {PKa}SIGNca {{Resp, N, M2}PKa}SIGNb
```

Pour vérifier que le message $M1$ est sûr c'est-à-dire qu'il reste confidentiel et que la réponse provient bien du nœud ayant reçu la requête, nous décrivons ce protocole dans un fichier AVISPA (donné en annexe C). Nous obtenons exactement le même résultat que pour les échanges de type 1, c'est-à-dire que pour les buts $secrecy\_of\_m$ et $authentication\_on\_alice\_bob\_n$ le protocole explicité dans le fichier $avispa\_exch2.if$ est $SAFE$. Les résultats sont les mêmes pour les trois autres modèles.

Echange de type 3 Dans ce type d'échange il s'agit simplement pour un nœud A d'envoyer dans une requête d'identifiant *Req* un message M devant rester confidentiel. Pour garder M secret, l'échange doit être chiffré en plus d'être signé. Le message étant adressé à tous les nœuds du groupe gr, il est chiffré avec la clé $PKgr$ de ce groupe.

```
1. A -> B : {PKa}SIGNca {{Req, N, M}PKgr}SIGNa
```

Pour vérifier que le message M est sûr c'est-à-dire qu'il reste confidentiel, nous décrivons ce protocole dans un fichier AVISPA (donné en annexe C). Le résultat est le même que précédemment, c'est-à-dire que pour le but $secrecy\_of\_m$ le protocole explicité dans le fichier $avispa\_exch3.if$ est $SAFE$ pour tous les modèles.

Remarque. Pour les échanges de type 1 et 2, une attaque spécifique aux réseaux sans-fil peut être menée. Les échanges seraient pour le type 1 :

```
a -> i : {PKa}SIGNca, {Req, N}SIGNa
i -> b : {PKa}SIGNca, {Req, N}SIGNa
b -> i : {Resp, N, M}PKa, SIGN(Resp, N, M)
b -> a : {Resp, N, M}PKa, SIGN(Resp, N, M)
```

On aurait de la même manière pour le type 2 :

```
a -> i : {PKa}SIGNca, {Req, N, M1}SIGNa
i -> b : {PKa}SIGNca, {Req, N, M1}SIGNa
b -> i : {Resp, N, M2}PKa, SIGN(Resp, N, M2)
i -> a : {Resp, N, M2}PKa, SIGN(Resp, N, M2)
```

On considère les échanges ayant lieu sur i qui représente l'attaquant ; a et b représentent respectivement *Alice* et *Bob*. a envoie une requête de découverte. i récupère cette requête et la transmet à b. b répond à i. Le secret est bien gardé (testé précédemment) car il est chiffré avec la clé publique de a. i retransmet la réponse de b vers a. a peut donc croire que b est dans le voisinage alors qu'il ne l'est pas. Nous verrons pour chaque protocole (présentés dans les sous sections d à j) les conséquences d'une telle attaque.

d) Validation de la découverte des groupes voisins

A (représentant Alice tout au long des validations) envoie une requête de découverte des groupes voisins de la forme :

$a; seqNum$

B (représentant Bob tout au long des validations renvoie une réponse de la forme :

$b; seqNum; group\_id; group\_key; type\_conn; challenge; response; type\_lead; description; type\_adm; nb\_adm; nb\_excl$

Les traces observées sont les suivantes.
Du côté de A :

log

```
envoi: a;2133977030
reception : b;2133977030;gr,idKey227299366,chal,question,n,lead,desc,b,
                                                                    1,1
```

Du côté de B :

log

```
reception : a;2133977030
envoi : b;2133977030;gr,idKey227299366,chal,question,n,lead,desc,b,1,1
```

On constate que l'échange se passe donc conformément à ce que l'on souhaitait.

Concernant la validation du protocole par AVISPA, nous sommes dans le cas d'une communication de type 1 décrite en c) avec M la liste des OPG qu'un nœud renvoie en réponse à la requête. Le protocole est donc valide.
L'attaque décrite elle aussi en c) n'a pas d'intérêt car elle conduit simplement i a servir de routeur entre les deux nœuds. Il pourrait seulement faire croire que b est dans le voisinage de a alors qu'il ne l'est pas.

e) Validation de la connexion à un groupe par vote

L'admission dans le groupe se fait par vote des éléments extérieurs ; A envoie une requête de connexion de la forme :

$c; seqNum; group\_key; ownerId; reponse$

B renvoie une réponse de la forme :

$d; seqNum; dst\_id; group\_key; vote$

Les traces observées sont les suivantes.
Du côté de A :

log

```
envoi: e;926075181;idKey227299366;alice,idKey1671322634;rep_alice
reception : f;926075181;alice,idKey1671322634;idKey227299366;o
```

Du côté de B :

log

```
reception : e;926075181;idKey227299366;bob,idKey1453756686;rep_alice
envoi: f;926075181;alice,idKey1671322634;idKey227299366;o
```

Les échanges se sont bien passés.

En utilisant AVISPA, nous vérifions ensuite que la réponse au *challenge* est bien gardée secrète. Nous souhaitons aussi l'authentification de la réponse à la requête en fonction du *nonce*. Nous sommes dans le cas d'une communication de type 1 décrite en c) avec M le vote de l'utilisateur qui a reçu la requête. Le protocole est donc validé.
L'attaque elle aussi décrite en c) n'a pas d'intérêt car a (représentant *Alice*) peut simplement recevoir le vote d'un membre qui n'est pas dans son voisinage immédiat, ce qui n'invalide en rien le résultat du vote.

f) Validation de la découverte des identités voisines d'un même groupe

A et B font maintenant partie du même groupe, ils peuvent se voir l'un l'autre lors de l'invocation d'un *inquiry* sur les identités voisines. A et B veulent découvrir les membres voisins pour le groupe considéré.
A et B envoient une requête de découverte des identités voisines d'un même groupe de la forme :

$c; seqNum; group\_key;$

A et B renvoient une réponse de la forme :

$d; seqNum; group\_key; ownerId$

Les traces observées sont les suivantes.
Du côté de A :

log

```
reception : c;622027874;idKey227299366
envoi: d;622027874;idKey227299366;alice,idKey1671322634

envoi: c;1313665888;idKey227299366
reception : d;1313665888;idKey227299366;bob,idKey1453756686
```

Du côté de B :

log

```
envoi: c;622027874;idKey227299366
reception : d;622027874;idKey227299366;alice,idKey1671322634

reception : c;1313665888;idKey227299366
envoi: d;1313665888;idKey227299366;bob,idKey1453756686
```

L'échange est conforme à ce qui est attendu.

A l'aide d'AVISPA, il faut vérifier l'authentification de la réponse à la requête par le *nonce*. Nous sommes dans le cas d'une communication de type 1 décrite en c) avec M l'identité du voisin qui a reçu la requête. Le protocole est donc validé.
Ici, l'attaque décrite elle aussi en c) a pour conséquence que A pourrait voir apparaître une identité qui n'est pas réellement dans le voisinage. L'attaquant n'en tirerait aucun avantage.

g) Validation de l'échange d'un élément secret entre deux noeuds

A envoie une requête de partage d'un secret de la forme :
$j; seqNum; dst\_key; group\_key; owner; secret$

B renvoie une réponse de la forme :
$k; seqNum; group\_key; owner\_key; vote$

Les traces observées sont les suivantes.
Du côté de A :

log

```
envoi: j;229403226;idKey1453756686;idKey227299366;alice,idKey1671322634;
                                                                    comm_ab
reception : k;229403226;idKey227299366;idKey1453756686;o
```

Du côté de B :

log

```
reception : j;229403226;idKey1453756686;idKey227299366;
                                                    alice,idKey1671322634;comm_ab
envoi: k;229403226;idKey227299366;idKey1453756686;o
```

Tout se passe donc comme nous le souhaitions.

En utilisant AVISPA, nous vérifions que le secret reste strictement confidentiel entre les deux éléments et que l'authentification de la requête par le *nonce* est possible. Nous sommes dans le cas d'une communication de type 2 décrite en c) avec $M1$ le secret à partager et $M2$ la réponse du récepteur. Le protocole est donc validé.
L'attaque décrite elle aussi en c) permet simplement à l'intrus i de jouer le rôle de routeur entre les deux éléments sans que cela ne lui apporte quoique ce soit.

h) Validation des communications de groupe

A envoie un message au groupe de la forme :
$l; group\_key; owner; message; clock; v\_leader$

Du côté de A :

log

```
envoi: l;idKey227299366;alice,idKey1453756686;bonjour;n;peter
```

Les traces observées sont les suivantes.
Du côté de A :

log

```
reception : l;idKey227299366;alice,idKey1453756686;bonjour;n;peter
```

L'échange se déroule conformément à ce que l'on attendait. Notons qu'un troisième nœud est entré dans le groupe et est devenu *leader* (on peut l'oberver dans le dernier paramètre du quadruplet échangé).

Pour ce qui concerne l'utilisation d'AVISPA, nous avons ici besoin de vérifier que le message est bien gardé secret entre les nœuds du groupe. Nous sommes dans le cas d'une communication de type 3 décrite en c) avec M le message adressé au groupe. Le protocole est donc validé.

i) Validation du mécanisme de plainte

B envoie une plainte de la forme :

$m; group\_key; dst\_key; owner\_key$

A ne renvoie aucune réponse.
Les traces observées sont les suivantes.
Du côté de B :

log

```
envoi : m;idKey227299366;idKey868822143;idKey1453756686
```

Du côté de P :

log

```
reception : m;idKey227299366;idKey868822143;idKey1453756686
```

La plainte est correcte et B l'a bien reçue. Le protocole, selon exactement le même principe que pour la communication de groupe, est sûr. La seule différence est la nature du secret : il ne s'agit plus d'un message à récupérer par les utilisateurs membres mais d'un vote ne nécessitant pas de réponse de la part de son récepteur.

j) Validations concernant le *Leader*

Modification d'un quadruplet. Rappelons tout d'abord qu'un quadruplet regroupe l'ensemble des paramètres d'un nœud dans l'algorithme d'élection du *leader*, i.e. son identité, l'identité du *leader*, le nombre de saut jusqu'au *leader*, le prédécesseur dans le chemin vers le *leader*.

Le quadruplet de B vient d'être modifié, il diffuse alors son nouvel état. A, qui connaît B comme voisin, reçoit ce quadruplet et envoie à son tour son actualisation. Notons que si B n'est pas reconnu comme voisin, A envoie simplement une requête de découverte des voisins (pour être sûr que B est accessible). Nous reviendrons sur les phases de l'algorithme un peu plus loin.
A et B s'échangent des messages d'actualisation, tant que c'est nécessaire. Ces messages sont du type :

$g; seqNum; group\_key; quad; state\_mile$

Les traces observées sont les suivantes.
Du côté de B :

log

```
envoi: g;146306338;idKey227299366;bob:bob:n:bob:bob:n;u
reception : g;146306338;idKey227299366;alice:bob:1:bob:alice:n;u
```

Du côté de A :

log

```
reception : g;146306338;idKey227299366;bob:bob:n:bob:bob:n;u
envoi: g;146306338;idKey227299366;alice:bob:1:bob:alice:n;u
```

Ici Bob (représenté par B) est devenu *leader* : quand Alice (représenté par A) reçoit le message, son quadruplet se met à jour et elle le renvoie à Bob. Les échanges s'arrêtent là puisque le quadruplet de Bob ne change pas à ce moment là. Notons que l'horloge de Bob a été adoptée par Alice.

Imaginons notamment qu'un nouveau nœud P apparaisse, de propriétaire Peter. Nous pouvons alors voir que A et B réceptionnent son quadruplet et renvoient leurs changements ; ils ont tous les deux adopté Peter et ne changent rien quand ils reçoivent le quadruplet l'un de l'autre.

Les traces observées sont les suivantes.
Du côté de B :

log

```
reception : g;1419035490;idKey227299366;peter:peter:n:peter:peter:n;u
envoi: g;1419035490;idKey227299366;bob:peter:1:peter:bob:n;u

reception : g;1419035490;idKey227299366;alice:peter:1:peter:alice:n;u
```

Du côté de A :

log

```
reception : g;1419035490;idKey227299366;peter:peter:n:peter:peter:n;u
envoi: g;1419035490;idKey227299366;alice:peter:1:peter:alice:n;u

reception : g;1419035490;idKey227299366;bob:peter:1:peter:bob:n;u
```

Tout c'est donc bien passé. La validation formelle de l'algorithme d'élection du *leader* est donnée dans l'annexe A (page 181).

Il suffirait ici de valider avec AVISPA l'authentification de la réponse à la requête par le *nonce*, exactement de la même manière que pour les points précédents (lors de la découverte des voisins par exemple). L'attaque décrite en c) est la même. Par contre dans ce cas précis des problèmes se posent : un attaquant i pourrait nous faire croire que l'un des nœuds b est dans le voisinage de a alors qu'il ne l'est pas. Il fausserait ainsi les chemins dans l'arbre (des plus courts chemins vers le *leader*). Cependant, la découverte du voisinage se faisant très régulièrement, il perdrait beaucoup d'énergie à maintenir cette erreur et ceci ne lui apporterait rien. S'il abandonne ce comportement « malveillant », l'algorithme reconstitue un arbre correct.

Découverte du voisinage. Tout au long de sa participation à un groupe, chaque noeud fait régulièrement une découverte de ses voisins pour les réactualiser. Chacun envoie une requête de découverte des quadruplets voisins de la forme :

$h; seqNum; group\_key$

Et chacun renvoie une réponse du type :
$i; seqNum; quad; group\_key$

Les traces observées sont les suivantes.
Du côté de B :

log

```
envoi: h;28761984;idKey227299366
reception : i;28761984;alice:bob:1:bob:alice:n;idKey227299366
```

Du côté de A :

log

```
reception : h;28761984;idKey227299366
envoi: i;28761984;alice:bob:1:bob:alice:n;idKey227299366
```

L'échange se passe normalement. Il n'y a aucun changement donc aucun autre quadruplet n'est renvoyé. Si c'était le cas il y aurait simplement un échange de plus. La validation AVISPA est la même que pour la modification d'un quadruplet.

Conclusion

Dans ce chapitre nous avons donc présenté et validé l'application SManet tant formellement que par des tests réels. Cependant d'autres tests à plus grande échelle doivent être réalisés.

Chapitre 5

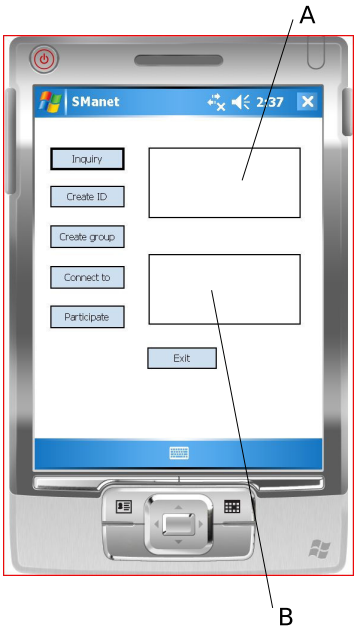
SManet : utilisation de l'application

Dans ce chapitre, nous détaillons le fonctionnement de l'interface graphique de l'application, puis nous décrivons des scénarios qui illustrent son utilisation.

5.1 Mode d'emploi de l'interface graphique

5.1.1 Fenêtre principale

La fenêtre principale de l'application est composée de deux parties. Sur la moitié droite en partie haute (A, fig. 65) se trouve une liste où s'afficheront les noms des groupes voisins, et en partie basse (B, fig. 65), une liste où apparaîtront les identités utilisateurs. Sur la moitié gauche des boutons permettent de réaliser les fonctions suivantes :



Inquiry : lancement de la découverte des groupes voisins ;

Create ID : création et enregistrement d'une nouvelle identité ;

Create Group : création et enregistrement d'un nouveau groupe ;

Connect to : inscription à un des groupes choisie dans la liste des groupes voisins ;

Participate : observation des paramètres de participation aux groupes auxquels on est connecté sous une certaine identité.

FIG. 65 – Fenêtre principale

5.1.2 Création et enregistrement d'identité

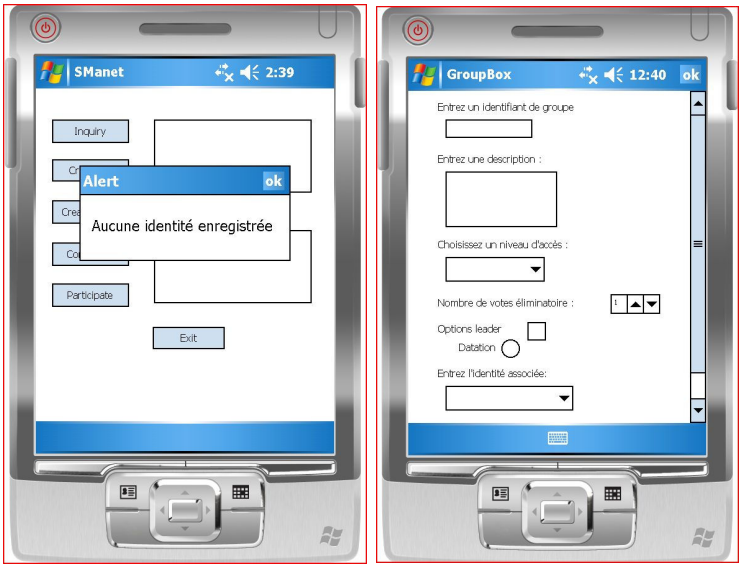
Pour créer une nouvelle identité il faut tout d'abord cliquer sur le bouton *Create ID*. Une nouvelle fenêtre apparaît (fig. 66(a)) dans laquelle l'utilisateur doit entrer son pseudonyme (fig. 66(b)). Il doit ensuite cliquer sur le bouton *ok* pour envoyer la demande de création à la carte. Si aucun pseudonyme n'a été entré, un message apparaît le prévenant qu'il faut fournir une identité. Sinon l'utilisateur se retrouve de nouveau sur la fenêtre principale, et l'identifiant nouvellement enregistré apparaît dans la liste inférieure (66(c)).

(a) Fenêtre *Create ID* (b) Entrée du pseudo (c) Retour à la fenêtre principale

FIG. 66 – Création et enregistrement d'une identité

5.1.3 Création et enregistrement d'un groupe

Pour participer à un groupe, il faut indiquer l'identité sous laquelle l'utilisateur va y participer (cela est aussi vrai pour le créateur du groupe). Ainsi, si un utilisateur clique sur le bouton *Create Group* alors qu'il n'a pas d'identité, un message d'erreur apparaît (fig. 67(a)). Au contraire, si l'utilisateur a déjà des identités, une nouvelle fenêtre apparaît dans laquelle se trouve une liste de champs à remplir (fig. 67(b)). Les trois premiers sont toujours les mêmes, à savoir le pseudonyme du groupe, sa description et le niveau de contrôle d'accès associé.



(a) Message d'erreur (b) Fenêtre *Create Group*

Fig. 67 – Création et enregistrement d'un groupe - 1

Deux niveaux d'accès sont possibles :

- *open* (fig. 68(a)) : tout utilisateur demandant à entrer sera automatiquement accepté sans aucune condition. Il reste alors à fournir les informations suivantes : le nombre de plaintes nécessaires pour évincer un individu ; si l'on souhaite faire fonctionner l'algorithme du *leader* et dans ce cas la tâche qu'il devra réaliser (ici il n'y a que la datation des messages) ; l'identité avec laquelle l'utilisateur participera au groupe ;

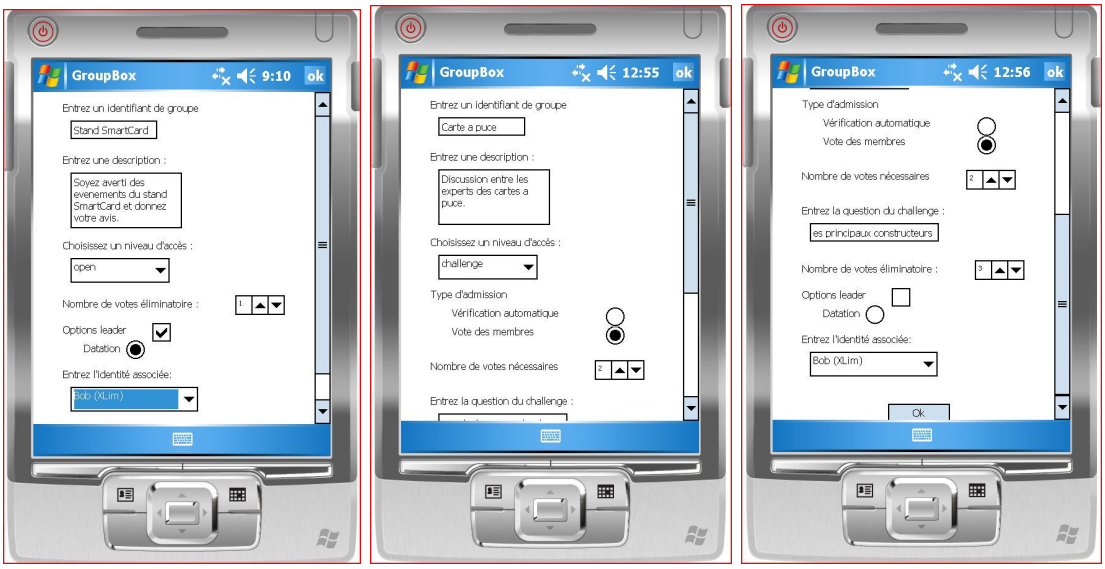
- *challenge* ou *private* (fig. 68(b), 68(c), 69(a) et 69(b)) : les utilisateurs doivent respecter une condition d'entrée. On retrouve les mêmes champs que pour le mode *open* mais d'autres s'ajoutent :
  - le champ indiquant si le mode d'admission est automatique (le ou les secrets sont contenus dans la carte) ou humain (la réponse de l'utilisateur est soumise au vote des membres) auquel cas un champ supplémentaire détermine le nombre de votes nécessaires pour qu'un utilisateur soit accepté (fig. 68(b) pour le mode *challenge* et 69(a) pour le mode *private*) ;

  - le champ où le créateur indique le *challenge* à résoudre pour entrer dans le groupe (une question par exemple, (fig. 68(c) pour le mode *challenge* et 69(b) pour le mode *private*) ;

  - si l'utilisateur a choisi un mode d'admission automatique, il y a un champ de plus où indiquer la ou les réponses possibles (séparées par le signe « + ») à donner par le candidat (voir fig. 69(b)) ;

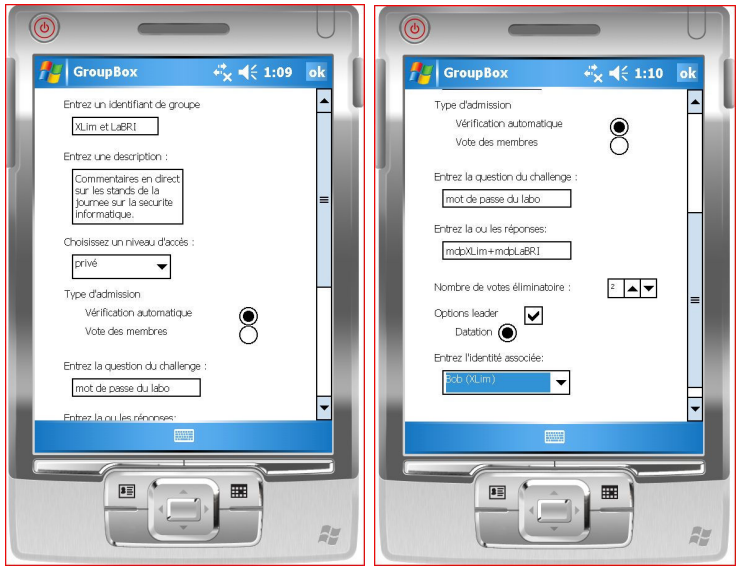
  Remarque : ces deux modes sont très proches. Ils ont un nom différent afin d'indiquer aux candidats potentiels si le groupe peut admettre n'importe qui par une question

ouverte (dont on peut connaître la réponse sans appartenir à un groupe humain particulier, mode *challenge*) ou par une question fermée (dont on ne peut connaître la réponse qu'en appartenant à un groupe humain particulier, mode *private*).



(a) mode *open* (b) mode *challenge* (c) mode *challenge*

FIG. 68 – Création et enregistrement d'un groupe - 2



(a) mode *private* - 1 (b) mode *private* - 2

FIG. 69 – Création et enregistrement d'un groupe - 3

5.1.4 Découverte des groupes voisins

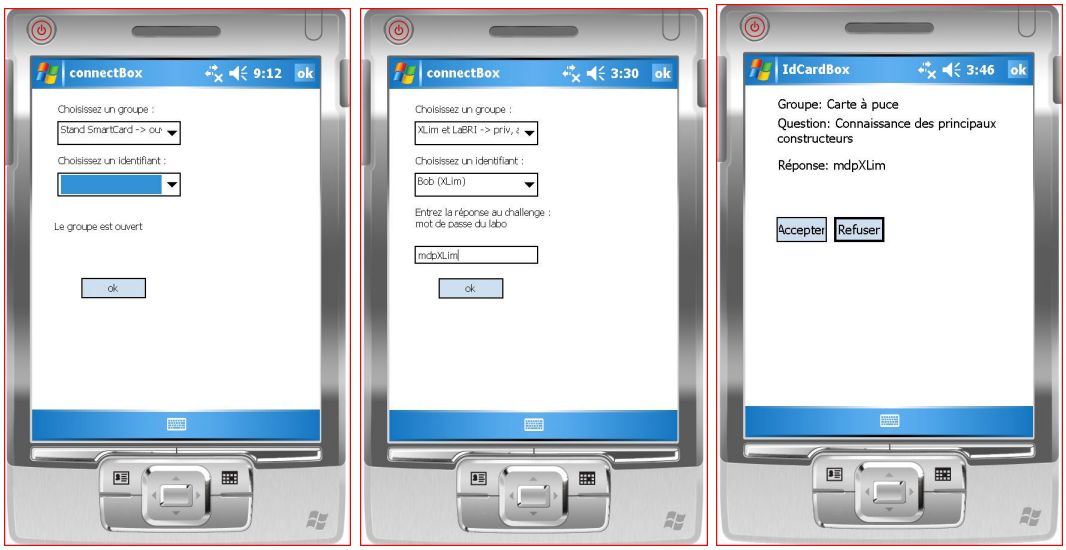
Le bouton *Inquiry* permet la découverte des groupes voisins. Ces derniers s'affichent dans la liste du haut (fig. 70(a)). On y voit l'identité du groupe, le niveau de contrôle d'accès, le mode d'admission, le cas échéant le nombre de votes positifs nécessaires pour accéder au groupe, le nombre de votes nécessaires pour évincer un membre. La description d'un groupe est accessible en cliquant sur la ligne où il est décrit (fig. 70(b) et 70(c)).

(a) Apparition des groupes (b) Description d'un groupe - 1 (c) Description d'un groupe - 2

Fig. 70 – Découverte des groupes voisins

5.1.5 Connexion à un groupe

Le bouton *Connect to* permet la connexion à un groupe. Dans la fenêtre qu'il ouvre, l'utilisateur doit choisir un groupe et une identité à lui associer. Si le groupe est ouvert, une phrase l'indique (fig. 71(a)) et il suffit de valider ses choix. Si le groupe concerné est à accès restreint, il apparaît en plus la question d'entrée et une zone d'édition de texte pour entrer la réponse (fig. 71(b)). Pour les admissions par vote, les autres membres reçoivent une demande de vote (fig. 71(c)).



(a) Groupe ouvert (b) Groupe privé (c) Vote d'admission

FIG. 71 – Connexion à un groupe

5.1.6 Participation à un groupe

Les groupes auxquels un utilisateur est connecté sont classés en fonction de l'identité qui leur a été associée. Ainsi, pour activer un groupe et observer les paramètres définissant la façon d'y participer, il faut sélectionner une identité dans la liste inférieure droite et cliquer sur *Participate*.

Fenêtre des groupes

La nouvelle fenêtre est découpée en autant de zone qu'il y a de groupes auxquels on est inscrit ou en cours d'admission (fig. 72(a)). Si on est inscrit au groupe la zone comprend le nom du groupe, la liste des identités des membres présents dans le voisinage et un ensemble de boutons. S'il manque des votes, on voit simplement affiché le nom du groupe et le nombre de votes actuel. (fig. 72(b), 72(c))

Activation d'un groupe

Une fois inscrit dans un groupe, l'utilisateur peut choisir d'y participer ou non (cf. 2.4.3, remarque 3 du paragraphe traitant de la gestion de l'autorisation d'admission, page 74). Pour activer un groupe il faut cliquer sur le bouton *Activate*. Si un groupe fait fonctionner l'algorithme du *leader*, le quadruplet correspondant s'affiche sous le bouton *Vote Against* (fig. 73).

(a) Groupe connecté (b) Admission en cours 1 (c) Admission en cours 2

Fig. 72 – Groupes associés à une identité

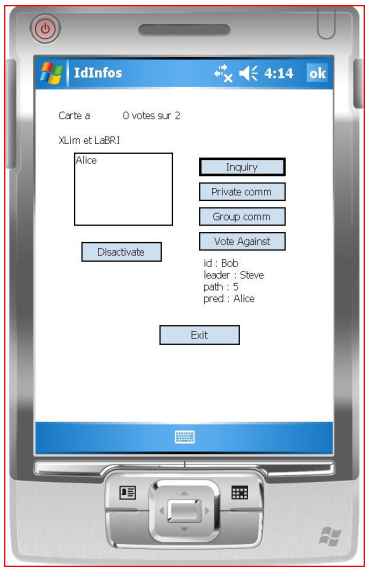


Fig. 73 – Activation d'un groupe

Inquiry et *leader*

Le bouton *Inquiry* sert à lancer la recherche des identités des membres voisins au sens du groupe. Ils apparaissent ensuite dans la liste associée (fig. 73).

(a) Echange d'un secret - 1 (b) Echange d'un secret - 2

FIG. 74 – Communications privées - échange d'un secret

Echange avec une identité particulière. Le bouton *Private comm* permet d'échanger un message avec un membre du groupe désigné par son identité. S'il s'agit d'un élément secret dont on souhaite qu'il soit conservé sur la carte du destinataire associé à l'identité de l'émetteur, afin que le partage de ce secret ne puisse pas être nié il faut cocher l'option « Rec » (fig. 74(a)). Une fois le secret entré et la validation faite, une fenêtre apparaît chez le récepteur (fig. 74(b)). Il a alors le choix d'accepter ou non le partage de cet élément. S'il l'accepte celui-ci est alors enregistré sur son dispositif.

Si le message à échanger est simplement un texte à lire par le récepteur, il suffit de laisser la case « Rec » non cochée (fig. 75(a)). Une fois le message entré et la validation faite, une fenêtre apparaît chez le récepteur (fig. 75(b)) contenant le message en question.

(a) Echange d'un secret - 1 (b) Echange d'un secret - 2

Fig. 75 – Communications privées - échange d'un message simple

Communication avec le groupe

Le bouton *Group comm* permet d'envoyer un message à l'ensemble des membres du groupe : de la même manière que précédemment, l'initiateur entre le message (fig. 76(b)) et tous les membres le reçoivent (fig. 76(a)).

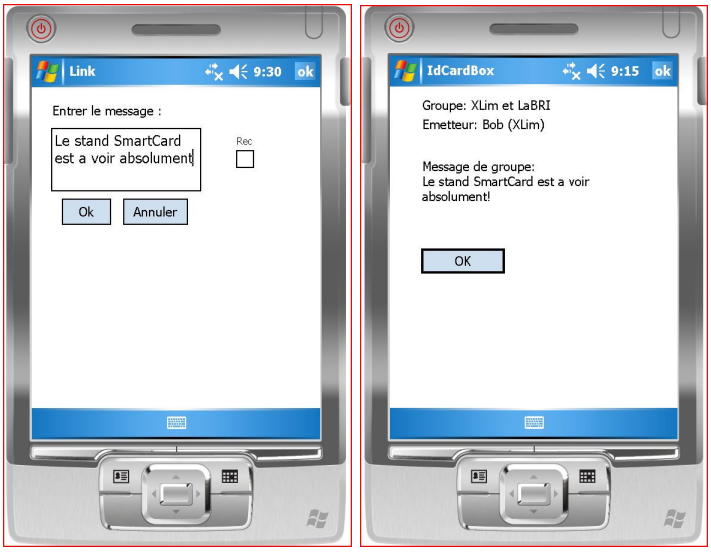


Fig. 76 – Communications de groupe

Eviction

Le bouton *Vote Against* permet de voter contre un élément qu'on aura sélectionné dans la liste. Lorsque le nœud ciblé reçoit le nombre de plaintes requis pour être banni du groupe, un message lui indique son éviction (fig. 77).

FIG. 77 – Exclusion d'un élément du groupe

5.2 Scénarios

Nous présentons ci-dessous des exemples d'utilisation de notre système pour chaque mode de groupe que nous avons mis en place.

5.2.1 Scénario de groupe fermé par un ensemble de secrets prédéterminés

Imaginons que les membres de deux laboratoires indépendants, donc deux organisations humaines différentes, se trouvent dans une conférence. Ils ont chacun de leur côté la capacité de former un groupe représentant leur organisation, en utilisant leur propre secret partagé. Imaginons encore que deux personnes appartenant à un laboratoire différent souhaitent créer un groupe unique réunissant les membres de leurs deux laboratoires afin de collaborer sur un projet. Avec une approche classique, il faudrait déterminer un secret commun et le diffuser à tous les membres potentiels par un canal d'échange sécurisé. Avec notre solution (fig. 78), ceci n'est plus nécessaire. Les deux membres initiaux se consultent (par un autre canal que le réseau) et entrent chacun leur secret dans le champ correspondant du formulaire de création d'un groupe (sous la forme $<secret1>+<secret2>$, voir fig. 69(a) et 69(b) page 152). En aucun cas les secrets ne seront révélés car protégés dans la carte à puce par l'OPG. A partir de là, l'OPG contenant les secrets pourra être récupéré par tous grâce à l'infrastructure d'échanges sécurisés globale formée par les cartes à puce. Ensuite, il suffira de connaître l'un des deux secrets pour être accepté comme membre du

groupe réunissant les deux groupes initiaux.

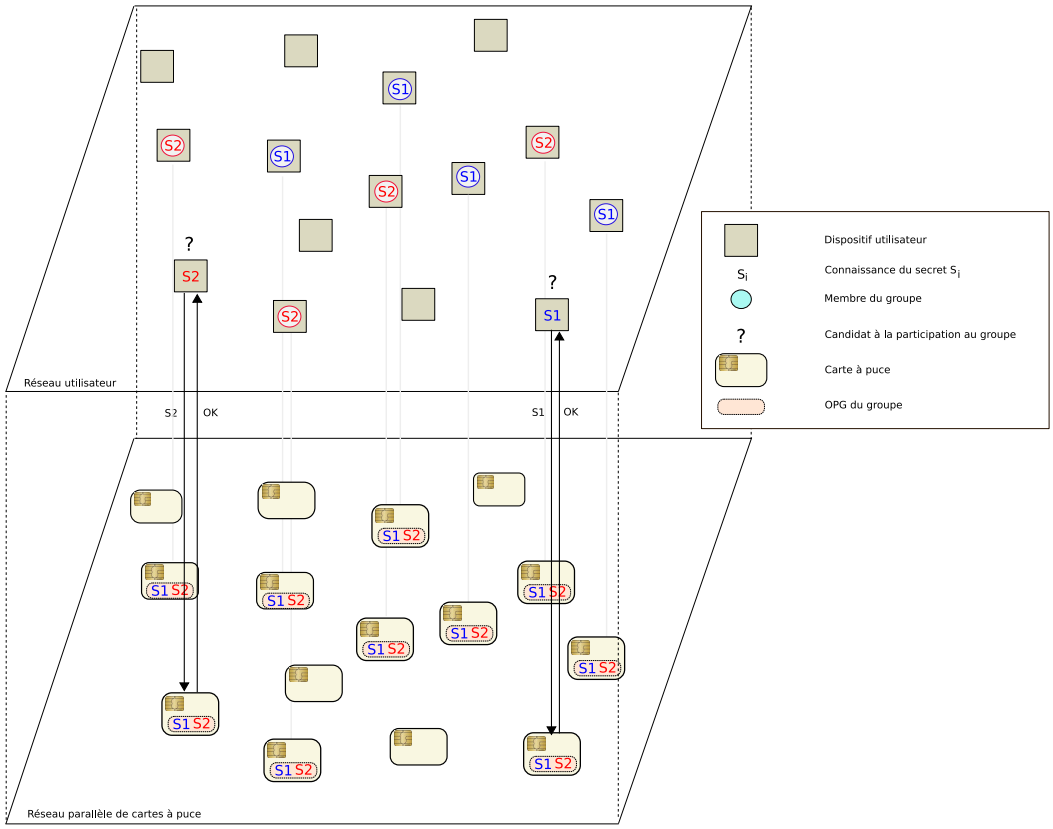


Fig. 78 – Exemple de constitution de groupes

5.2.2 Scénario de groupe fermé par vote (condition d'entrée faible)

Des individus ne faisant pas partie d'une organisation commune peuvent souhaiter participer à un groupe dont l'accès est restreint aux entités possédant des connaissances ou des capacités répondant au *challenge* défini par le créateur. Les connaissances annoncées par une entité voulant rejoindre le groupe seront jugées suffisantes ou non pour participer par les autres membres de ce groupe.

Par exemple, dans le cadre des journées de la sécurité informatique, un expert en cartes à puce souhaite partager ces impressions avec d'autres experts pendant qu'il visite les stands. Il créé donc un groupe portant sur les cartes à puce (voir fig 68(b) et 68(c) page 152). Un candidat à l'entrée dans le groupe concerné se verra poser une question en raport avec ce thème. La qualité des réponses des candidat permettra au membres déjà présents de juger du degré de ces connaissance en la matière. En fonction de cela, ils choisiront de l'accepter ou non.

5.2.3 Scénario de groupe fermé par une condition d'entrée forte

Un groupe pourrait être restreint à des utilisateurs possédant une preuve d'authentification particulière telle qu'une carte bancaire. Le fait de posséder une carte bancaire peut être vérifié avec un lecteur de carte approprié. Au moment de l'authentification pour s'inscrire au groupe concerné, un code spécifique travaillerait en collaboration avec l'*applet* principale pour vérifier qu'une carte bancaire valide se trouve dans le lecteur. On pourrait ainsi obtenir une authentification forte avec l'identité réelle des participants. Une telle condition d'entrée permettrait d'encourager la mise en place de scénarios de services de haut niveau.

5.2.4 Scénario de groupe ouvert

Imaginons que des individus soient réunis dans une même zone mais qu'ils n'aient aucune interaction hors du réseau. L'exemple souvent utilisé dans ce cas est celui de l'embouteillage où des conducteurs pourraient former, selon leur intérêt ponctuel (par exemple leur lieu de destination) un réseau de partage de renseignements sur les conditions de circulation et les évènements en cours. Pour une utilisation plus commerciale, des boutiques pourraient créer un groupe dans lequel elles diffuseraient par inondation du réseau des informations sur leurs produits, touchant ainsi un cercle plus étendu que les personnes présentes dans le magasin.

Si on reprend l'exemple des journées de la sécurité informatique, un industriel présentant un stand peut créer un groupe ouvert proposant aux utilisateurs de les informer des évènements du stands tout au long de la journée et d'échanger leurs commentaires (voir fig. 68(a) page 152).

Enfin, pour une utilisation dans le domaine des loisirs, tout partage d'information comme des photos, des vidéos ou des renseignements sur un lieu, peut faire l'objet d'un groupe ouvert. On notera que la fourniture d'une mauvaise information peut alors faire l'objet d'une plainte grâce au mécanisme d'identité unique que nous avons mise en place.

Conclusion

L'implémentation de l'application SManet utilise de nombreuses technologies logicielles et matérielles. Les cartes à puce sont des dispositifs disposant de très peu de ressources et en particulier très peu de mémoire. De part cette contrainte nous n'avons pas pu tester notre implémentation avec de vraies cartes à puce. Nous avons donc réalisé une émulation quasiment identique du point de vue du code ce qui nous a permis de faire tous les tests nécessaires. L'ensemble des fonctionnalités est opérationnel et a été formellement validé.

Conclusion

Notre objectif dans cette thèse était de permettre l'établissement entièrement spontané de communications sécurisées (« n'importe où, n'importe quand, entre n'importe qui et n'importe qui ») et d'aller jusqu'à l'établissement de groupes d'entités partageant un même but, qu'ils requièrent ou non un accès sécurisés. Les solutions existantes ne nous convenaient pas car elles posent des contraintes faisant obstacle à la spontanéité que nous souhaitons, se basant sur des planifications ou des interactions humaines. Nous avons donc dû pour cela reprendre un certain nombre de notions et de définitions classiques pour les adapter et/ou les préciser dans notre contexte particulier des MANets purs réellement mobiles.

Résumé des contributions

Par cette thèse, nous contribuons à rendre flexible l'utilisation des réseaux MANets tout en rajoutant de la sécurité. La contrepartie est l'utilisation obligatoire d'une carte à puce. Bien que toutes les cartes du système doivent posséder des données de base identiques (une application et les clés associées), elles sont indépendantes, aussi bien dans leur préparation que dans leur déploiement, de toute organisation pouvant gérer des groupes d'utilisateurs : les utilisateurs ne partagent pas obligatoirement de lien (pas d'organisation commune) ni de secret. Ils ne se concertent pas non plus en parallèle du réseau. La création d'un groupe et la participation à ce groupe se font de manière entièrement dynamique, spontanée et décentralisée.
Ces travaux ont donné lieu à des publications [9, 8, 10] et à la présente thèse [7].

Infrastructure de cartes à puce

Le système est sécurisé par l'infrastructure de cartes à puce que nous avons mise en place. Chaque utilisateur dispose de sa propre carte, qui a pour rôle de contrôler les différentes étapes des protocoles de gestion des identités et de gestion des accès aux groupes supportés par notre système. Les cartes à puce offrent une administration (elle-même sûre) de la sécurité à l'échelle du réseau tout en restant locale à chaque nœud. Cette infrastructure peut être utilisée pour le contrôle de n'importe quel protocole et pourrait donc être exploitée dans d'autres applications.

Gestion des identités

Dans notre solution, les identités possèdent des propriétés garanties : elles sont uniques, non répudiables et une seule est utilisable pour un groupe donné. Les communications entre

utilisateurs ciblés (grâce à ces identités) se font au sein de groupes. Ces groupes justifient ces échanges et leur offrent un cadre. Ils sont décidés et créés à l'intérieur même de l'infrastructure de cartes à puce. Ainsi, notre solution permet de se servir du réseau ad hoc pour définir spontanément des groupes sécurisés sans utiliser d'autre moyen extérieur au réseau (ni concertation, ni échange physique de secret, etc.).

Les cartes à puce nous permettent de protéger la génération, le stockage et la distribution des identités des dispositifs et des groupes. Elles nous permettent aussi de protéger l'unicité de ces identités en interdisant leur manipulation (consultation ou échange) au niveau utilisateur. En effet, l'ensemble de ce processus est effectué en local. Une méthode de la carte génère (en local) un secret unique par nature et l'associe à un identifiant choisi par l'utilisateur. L'identité est alors liée à une carte sans possibilité pour un utilisateur de recréer la même identité sur une autre carte. Cette unicité est essentielle pour éviter tout conflit ou vol d'identité. Notre système permet aussi que chaque identité soit stockée de manière permanente sur les cartes, ce qui les rend non-répudiables. Nous garantissons également qu'une seule identité est utilisable par un même dispositif pour un même groupe.

Le contrôle total assuré par les cartes à puce est le fondement nécessaire qui permet d'apporter la confiance entre les utilisateurs, chaque utilisateur qui communique avec un autre possédant la garantie que cet autre utilisateur a obtenu une identité suivant la même procédure et respectant les mêmes propriétés que lui.

Ceci permet par ailleurs l'identification sans faille des éventuels nœuds malveillants. C'est une base essentielle pour la traçabilité et donc la sécurité vis-à-vis des comportements des utilisateurs.

Formation de groupes

Comme nous l'avons déjà écrit ci-dessus, les groupes donnent un cadre aux communications et constituent la raison des échanges. C'est pourquoi nous avons établi une solution de formation de groupes spontanée, i.e. sans planification. Elle offre un système de publication des caractéristiques des groupes pour qu'ils soient visibles auprès d'éventuels candidats : elle met en œuvre de manière sécurisée un système de gestion des accès à ces groupes ne nécessitant aucune entité extérieure et qui est distribué sur chaque nœud.

Publication des caractéristiques d'un groupe. Les utilisateurs choisissent d'entrer dans un groupe en prenant connaissance de ses caractéristiques. La publication de ces caractéristiques se fait par distribution, sur le réseau sécurisé de cartes à puce, d'un élément que nous appelons « OPG » (Objet Personnel de Groupe). Cet objet contient des informations que les cartes à puce utiliseront pour gérer l'accès au groupe. Une partie de ces informations est communiquée à l'utilisateur de la carte afin de décrire le groupe auquel il pourrait avoir accès. L'autre partie des informations reste confidentielle au sein de cette carte. Ce sont les éléments qui vont permettre d'organiser la procédure d'accès. Il s'agit en particulier de ce que va devoir fournir un candidat à l'entrée dans un groupe. Cela peut

être par exemple la réponse à une question. Il peut aussi s'agir d'un lien vers un mécanisme de vérification plus complexe (possession d'une carte bleue, d'un type de dispositif particulier, d'une application privée, etc), voire une méthode *Two Factor Authentication*. Cette approche permet de former des groupes de manière spontanée et totalement distribuée puisque toute la procédure de validation d'accès est réalisée en local, sans qu'il y ait besoin de l'intervention d'une entité extérieure.

Notre solution assure à tous les éléments présents dans un groupe que tous les autres éléments également présents se sont eux aussi pliés à la procédure d'accès et que la réussite de cette procédure leur a permis d'entrer dans le groupe. Ainsi, on établit une confiance partagée entre tous les membres d'un groupe au travers d'une procédure répartie sur chacun.

Gestion des accès à un groupe. La gestion des accès à un groupe peut être faite suivant plusieurs niveaux de restriction et des modes d'admission variés. Le groupe peut être à accès ouvert : dans ce cas l'admission est automatique c'est-à-dire que la carte autorise son utilisateur à participer au réseau sans condition d'authentification. L'accès à un groupe peut être restreint par une condition d'entrée : dans notre implémentation actuelle il s'agit de donner la réponse à une question. Il est possible d'ajouter des modules pour vérifier des conditions d'entrée plus complexes telle que la possession d'une carte bleue par exemple. Dans tous les cas, la carte que nous associons au dispositif mobile peut contrôler l'exactitude de la réponse si celle-ci ou la procédure permettant de le faire est contenue dans l'OPG. Sinon, et c'est une approche que nous avons aussi implémentée, la réponse peut être soumise au vote des autres membres du groupe ; ce sont eux qui jugent alors de la validité de la réponse. Notons qu'un utilisateur peut appartenir à plusieurs groupes. Enfin, notre solution permet de proposer un système de révocation par vote des membres du groupe pour exclure un utilisateur (de manière à ce qu'il ne puisse plus lire les messages du groupe).

L'évolution des MANets au cours du temps étant très forte, nous avons introduit la possibilité d'en tenir compte pour la sécurité des échanges. La protection des échanges antérieurs à l'entrée d'un membre dans le groupe (i.e. le fait qu' un nouveau membre ne puisse pas accéder aux messages précédant son arrivée) est réalisée à l'aide d'un algorithme d'élection d'un *leader*. Le *leader* a pour rôle de « dater » les messages. Un membre admis dans un groupe à une date d a seulement accès aux messages postérieurs à d. Cet algorithme résiste aux apparitions et disparitions de nœuds ainsi qu'au partitionnement des groupes. Il est également conçu de manière à ce que la charge supplémentaire pour le leader soit minimale.

Bilan et perspectives

Bilan

Alors que les recherches concernant les réseaux MANets sont de plus en plus nombreuses, on voit finalement très peu d'applications réellement mises en place et utilisées. Nous avons vu que jusqu'à présent les applications existant pour ces réseaux permettaient de les utiliser de manière plus ou moins sûre mais en imposant toujours des contraintes aux

utilisateurs telles qu'une faible mobilité, l'utilisation d'un serveur central qu'il faut localiser et auquel il faut pouvoir accéder, une préparation des nœuds, etc. De plus, dans tous les cas, une organisation humaine, qui justifie les communications et fournit les informations initiales pour la sécurité des échanges, est nécessaire. Ces contraintes restreignent ainsi beaucoup l'emploi des réseaux MANets purs et empêchent leur extension à des situations plus spontanées.

Avec l'application SManet, nous proposons au contraire une solution utilisable dans n'importe quel contexte et pour n'importe quel niveau de sécurité, du plus fort (pour des services d'urgence par exemple) au plus faible (pour un rassemblement de personnes sur un sujet non sensible). Nous donnons aux utilisateurs un moyen d'organiser des groupes d'intérêt et d'assurer une sécurité suffisamment forte pour au moins permettre à ces groupes de conserver leur cohérence tout au long de leur vie, et ce en éliminant toute intervention et tout besoin d'interaction au niveau humain (après achat de la carte à puce). Pour cela nous avons mis en place une gestion sûre des mécanismes de sécurité suivants au sein des groupes : autorisation d'entrée ; authentification et admission sécurisée ; révocation des utilisateurs détectés comme malveillants par les autres membres ; etc. Il ne s'agit pas uniquement d'assurer la confidentialité des messages (qui ne le nécessitent pas toujours) mais également de garantir le « bon comportement » des membres du groupe, ce qui est essentiel pour qu'un groupe puisse fonctionner normalement. L'apport principal de l'application SManet est donc de réaliser localement sur chaque nœud toutes ces opération de sécurité et de mise en relation des utilisateurs qui habituellement étaient effectuées grâce à l'intervention d'une organisation humaine. Ainsi, la spontanéité n'est plus uniquement dans l'établissement matériel et logiciel du réseau, mais aussi dans la formation sécurisée d'un groupe partageant un intérêt commun. Au travers des scénarios que nous proposons, nous pouvons nous rendre compte de la diversité des contextes d'utilisation de l'application (utilisation professionnelle, commerciale, pour les loisirs, etc.).

La solution SManet a été mise en place pour les réseaux MANets mais elle peut aussi être utilisée dans n'importe quels type de réseau. Par exemple, sur Internet cela permettrait de mettre en place des applications peer-to-peer sans centralisation sur un serveur, entité qui était jusqu'à présent obligatoire au moins pour la mise en relation des différents participants. Avec l'application SManet il n'y aurait plus de problème de gestion sûre des identités.

Limites et perspectives

Pour la gestion des identités nous avons choisi de créer des identités uniques par nature qui ne peuvent être générées qu'une seule fois. Si une carte à puce est perdue, endommagée ou volée il sera impossible pour un utilisateur de recréer les mêmes identités que celles qu'il avait délivrées précédemment. Une approche différente serait de générer les identités à partir d'éléments par l'OPG (alors appelé charte) et des informations d'authentification fournies par l'utilisateur. Il serait alors possible de regénérer une identité particulière à partir des mêmes éléments, et ce sur n'importe quelle carte à puce. Avec cette solution, il faudrait accepter que la même identité puisse être possédée par plusieurs dispositifs.

Pour la gestion des groupes nous avons dans ce travail donné des exemples de conditions d'entrée simples, comme la réponse à une question. Pour la suite, nous souhaitons apporter plus de possibilités en ce qui concerne les procédures d'authentification et pour cela étudier le moyen de transporter du code dans les OPG eux-mêmes. Ce code pourrait être directement exécuté par les cartes réceptrices, sans intervention des hôtes. Ainsi, on chargerait la carte de réaliser une tâche particulière comme par exemple vérifier que l'utilisateur possède certaines ressources ou données, des capacités spécifiques, etc. La procédure d'entrée pourrait alors être entièrement modulable et déterminée par le créateur du groupe. Grâce à la sécurité des JavaCard les autres applications de la carte seraient protégées.

Nous avons vu que pour assurer la *backward secrecy*, l'algorithme d'élection du *leader* avait une faiblesse due à l'asynchronisme de la propagation des messages de mise à jour de la date à l'ensemble des éléments d'un même îlot. Nous avons considéré cette faiblesse comme raisonnable car les réseaux que nous considérons sont a priori petits et avec une courte durée de vie. Nous avons aussi vu que la technologie sans fil peut être non fiable et perdre des paquets, empêchant la détection d'un voisin potentiellement important. Cependant, nous avons expliqué que cela ne provoquait pas de conséquences graves sur l'algorithme d'élection de *leader* puisque, soit l'erreur est réparée à l'élection suivante soit le lien avec ce voisin n'est pas suffisamment fiable pour être pris en compte. Le problème deviendrait important si une erreur de détection d'un nœud voisin important (par exemple le *leader*) se produisait trop fréquemment. En effet, les nœuds passeraient alors leur temps à changer d'état. Le problème serait le même si ce voisin entrait et sortait réellement du voisinage concerné. Nous avons prouvé que cela n'empêche pas l'algorithme de tendre vers un état correct mais il est nécessaire de procéder à des tests réels à grande échelle pour mesurer en pratique les conséquences d'une telle situation.

La limite la plus importante de l'application SManet est celle due aux ressources des cartes à puce. En effet, ces cartes ont peu de mémoire et cela ne permet donc pas par exemple de pouvoir participer à un grand nombre de groupes. Cependant, dans un avenir proche les cartes auront beaucoup plus de ressources. De plus, des systèmes de partage de la mémoire du dispositif hôte se développent comme celui décrit en [43]. En attendant, cela soulève une question sur la possibilité de supprimer ou pas des groupes auquels on ne souhaite plus participer pour libérer l'emplacement mémoire qu'ils utilisent. Les identités ne sont en aucun cas effacées (elles sont stockées indépendemment du groupe) donc la non répudiation ne serait pas mise en danger. Cependant, après une éviction d'un groupe, si tous les paramètres du groupes sont effacés, un utilisateur pourrait y postuler de nouveau, sous une autre identité sans que quiconque sache qu'il vient d'être banni. L'utilisateur pourrait alors profiter de nouveau du groupe et éventuellement recommencer ses actes de malveillance. Dans ce travail nous avons choisi de ne pas implémenter la possibilité de supprimer les groupes pour éviter ce problème et conserver une plus grande sécurité, cependant il serait possible d'en faire une option déterminée lors de la création du groupe.

Si actuellement l'utilisation d'une carte à puce peut paraître contraignante, avec l'évolution rapide des dispositifs, il est probable que dans le futur tout équipement sera doté d'un lecteur de cartes à puce (c'est actuellement le cas de la plupart des PC portables). L'emploi de la carte à puce n'est donc pas une limitation à notre solution.

Dans la suite de ces travaux, notre premier objectif est un déploiement en vraie grandeur sur une soixantaine de PDA équipés de lecteurs de cartes à puce. Cela permettra

de valider opérationnellement notre solution. Dans un second temps, il s'agira d'étendre les fonctionnalités de l'application notamment pour la mise en place de moyens d'authentification plus complexes (vérification de la possession d'une carte bancaire par exemple) et d'étudier la possibilité de rendre la procédure d'authentification entièrement modulable afin que le créateur d'un groupe puisse choisir n'importe quel moyen de contrôle d'entrée.

Enfin, il serait intéressant que l'architecture développée puisse continuer à évoluer et que ces recheches se déroulent dans le cadre d'une collaboration avec des industriels.

Bibliographie

[1] J. N. Al-Karaki and A. E. Kamal. Routing techniques in wireless sensor networks : a survey. In IEEE Wireless Communications, volume 11, pages 6–28, 2004.

[2] S. Al-Riyami and K. Paterson. Certificateless public key cryptography. Technical report, Royal Holloway, University of London, 2003. http ://e-print.iacr.org/2003/126.ps.

[3] E. Althouse. Extending the littoral battlespace (ELB). advanced concept technology demonstration (ACTD). In Proceedings of the NATO Information Systems Technology Panel Symposium on Tactical Mobile Communications, juin 1999.

[4] G. Anastasi, E. Borgia, M. Conti, and E. Gregori. IEEE 802.11 ad hoc networks : Performance measurements. In Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCSW'03), page 758, Washington, DC, USA, 2003. IEEE Computer Society.

[5] G. Andreadis. Providing internet access to mobile ad hoc networks, 2002. http ://www.ee.ucl.ac.uk/lcs/papers2002/LCS120.pdf.

[6] A. Arnes. Public key certificate revocation schemes. Master's thesis, Norwegian University of Science and Technology, Department of Telematics, 2000. http ://www.pvv.ntnu.no/ andrearn/thesis/CertRevThesis_29Feb2000.ps.

[7] E. Atallah. Une solution pour l'établissement non planifié de groupes sécurisés permettant des communications sûres dans les réseaux MANets purs. PhD thesis, Laboratoires XLIM & LaBRI, 2008.

[8] E. Atallah, P.F. Bonnefoi, C. Burgod, and D. Sauveron. Mobile ad hoc network with embedded secure system. In Proceedings of the 1st International Conference Ambient Intelligence Developments 2006 (AmI.d2006), septembre 2006.

[9] E. Atallah and S. Chaumette. A smart card based distributed identity management infrastructure for mobile ad hoc networks. In 1st International Workshop In Information Security Theories and Practicies (WISTP'07), Heraklion, Greece, pages 1–13, mai 2007.

[10] E. Atallah, F. Darrigade, S. Chaumette, A. Karray, and D. Sauveron. A grid of java cards to deal with security demanding application domains. In 6th edition e-Smart conference & demos, Sophia Antipolis, Frensh Riviera, septembre 2005. e-smart 2005 Isabelle Attali Award for the best innovative technology.

[11] AVISPA v1.1 User Manual. www.avispa-project.org.

[12] E. Baccelli and P. Jacquet. Diffusion mechanisms for multimedia broadcasting in mobile ad hoc networks. In Proceedings of the IASTED International Conference on

Internet and Multimedia Systems and Applications (IMSA'04), Hawaii, USA, août 2004.

[13] A. Ballardie. Scalable multicast key distribution, 1996. ftp ://ftp.rfc-editor.org/in-urls/rfc1949.txt.

[14] T. Ballardie, B. Chain, and Z. Zhang. Core based trees (CBT) multicast routing architecture, septembre 1997. ftp ://ftp.rfc-editor.org/in-urls/rfc2201.txt.

[15] R. Barr. SWANS - scalable wireless ad hoc network simulator user's guide, 19 mars 2004.

[16] S. Basagni, K. Herrin, D. Bruschi, and E. Rosti. Secure pebblenets. In Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing (MobiHoc '01), pages 156–163, New York, NY, USA, 2001. ACM.

[17] M.C. Bañuls. Group Membership Protocols for Dynamic Environments. PhD thesis, Department of Information Systems and Computation, Universidad Politécnica de Valencia, Valencia , Spain, 2006. http ://www.dsic.upv.es/docs/bib-dig/tesis/etd-11082005-112051/thesismc.pdf.

[18] K. Becker and U. Wille. Communication complexity of group key distribution. In Proceedings of the 5th ACM conference on Computer and communications security (CCS'98), pages 1–6, New York, NY, USA, 1998. ACM Press.

[19] P. Bellavista, A. Corradi, and E. Magistretti. Proxy-based middleware for service continuity in mobile ad hoc networks. In Proceeding of the 4th Italian Workshop From Object To Agents : Intelligent Systems and Pervasive Computing (WOA'03), 2003.

[20] B. Bhattacharjee. Scalable secure group communication over ip multicast. In Proceedings of the Ninth International Conference on Network Protocols (ICNP'01), pages 261–269, Washington, DC, USA, 2001. IEEE Computer Society.

[21] L. Blazevic, L. Buttyan, S. Capkun, S. Giordano, J. Hubaux, and J. Le Boudec. Self-organization in mobile ad-hoc networks : the approach of terminodes. IEEE Communications Magazine, pages 166–174, 2001.

[22] R. Blom. An optimal class of symmetric key generation systems. In Proceeding of the EUROCRYPT 84 workshop on Advances in cryptology : theory and application of cryptographic techniques, pages 335–338, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[23] C. Blundo, A. De Santis, U. Vaccaro, A. Herzberg, S. Kutten, and M. Yung. Perfectly secure key distribution for dynamic conferences. Inf. Comput., 146 :1–23, 1998.

[24] Carlo Blundo, Alfredo De Santis, Ugo Vaccaro, Amir Herzberg, Shay Kutten, and Moti Yong. Perfectly secure key distribution for dynamic conferences. Inf. Comput., 146(1) :1–23, 1998.

[25] D. Boneh and M. Franklin. Efficient generation of shared RSA keys. In Proceedings of Advances in Cryptology (CRYPTO'97), pages 425–439, New York, NY, USA, 1998. Springer-Verlag.

[26] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the weil pairing. In Proceedings of the 7th International Conference on the Theory and Application of

Cryptology and Information Security (ASIACRYPT'01), pages 514–532, London, UK, 2001. Springer-Verlag.

[27] Dan Boneh and Matthew Franklin. Identity-based encryption from the weil pairing. SIAM J. Comput., 32(3) :586–615, 2003.

[28] J. Y. Le Boudec. Understand the simulation of mobility models with palm calculus. Technical Report Technical Report EPFL/IC/2005/033, EPFL-DI-ICA, EPFL, Lausanne, Switzerland, 2005. http ://ica1www.epfl.ch/perfeval/slides/leb-perf05.pdf.

[29] C. Boyd. On key agreement and conference key agreement. In Proceedings of the Second Australasian Conference on Information Security and Privacy (ACISP'97), pages 294–302, London, UK, 1997. Springer-Verlag.

[30] Jennifer Bray and Charles Sturman. Bluetooth : Connect Without Cables. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000.

[31] B. Briscoe. MARKS : Zero side effect multicast key management using arbitrarily revealed key sequences. In Proceedings of the First International COST264 Workshop on Networked Group Communication (NGC'99), pages 301–320, London, UK, 1999. Springer-Verlag.

[32] M. Burmester and Y. Desmedt. A secure and efficient conference key distribution system. In Advances in Cryptology - EUROCRYPT'94, pages 275–286, mai 1994.

[33] J. Cai and D.J. Goodman. General packet radio service in GSM. IEEE Communications Magazine, 35(10) :122–131, octobre 1997.

[34] T. Camp, L. Wilcox, and J. Boleng. Location information services in mobile ad hoc networks, 2002. http ://users.ices.utexas.edu/ lucasw/preprints/CampBoleng-Wilcox2002.pdf.

[35] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas. Multicast security : A taxonomy and some efficient constructions. In Proceeding of IEEE INFOCOMM, pages 708–716, 1999.

[36] R. Canetti and A. Herzberg. Maintaining security in the presence of transient faults. In Proceedings of the 14th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO'94), pages 425–438, London, UK, 1994. Springer-Verlag.

[37] S. Capkun, J.P. Hubaux, and L. Buttyan. Mobility helps security in ad hoc networks. In Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing (MobiHoc '03), pages 46–56, New York, NY, USA, 2003. ACM.

[38] Arnaud Casteigts. Model driven capabilities of the DA-GRS model. In Proceedings of the International Conference on Autonomic and Autonomous Systems (ICAS'06), page 24, Washington, DC, USA, 2006. IEEE Computer Society.

[39] Arnaud Casteigts and Serge Chaumette. Dynamicity aware graph relabeling systems (DA-GRS) - a local computation-based model to describe MANet algorithms. In Proceedings of the International Conference on Parallel and Distributed Computing and Systems (PDCS'05), Phoenix, USA, 2005. ACTA Press.

[40] Y. Challal and H. Seba. Group key management protocols : A novel taxonomy. IJIT International Journal of Information Technology, 2(2) :105–118, 2006.

[41] A. Chan and E. Rogers. Distributed symmetric key management for mobile ad-hoc networks. In Proceedings of the 23rd Conference of the IEEE Communications Society, IEEE Infocom 2004, 2004.

[42] X. Chang. Network simulations with OPNET. In Proceedings of the 31st conference on Winter simulation (WSC'99), pages 307–314, New York, NY, USA, 1999. ACM Press.

[43] S. Chaumette, A. Karray, and D. Sauveron. Secure extended memory for java cards. In Proceeding of the 11th IEEE Nordic Workshop on Secure IT-systems : NordSec 2006, Linköping, Suède, octobre 2006. Springer-Verlag.

[44] Z. Chen. Java Card Technology for Smart Cards. Addison-Wesley, 2000.

[45] B. Chor, S. Goldwasser, S. Micali, and B Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In Proceedings of the 26th Annual Symposium on Foundations of Computer Science (FOCS '85), pages 383–395, 1985.

[46] M. Lewis (Cisco). What is a virtual private network ? http ://www.ciscopress.com/content/images/1587051796/samplechapter/15870517-96content.pdf.

[47] T. Clausen and P. Jacquet. Optimized link state routing protocol (OLSR), 2003. http ://www.rfc-archive.org/getrfc.php ?rfc=3626.

[48] Y. Desmedt and Y. Frankel. Shared generation of authenticators and signatures (extended abstract). In Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO '91), pages 457–469, London, UK, 1992. Springer-Verlag.

[49] W. Diffie and M. Hellman. New directions in cryptography. In IEEE Transactions on information Theory, volume IT-22, pages 644–654, novembre 1976.

[50] L. Dondeti, S. Mukherjee, and A. Samal. Scalable secure one-to-many group communication using dual encryption, 1999. http ://citeseerx.ist.psu.edu/viewdoc/download ?doi=10.1.1.25.7787&rep=rep1&type=pdf.

[51] L. R. Dondeti, S. Mukherjee, and A. Samal. A dual encryption protocol for scalable secure multicasting. In Proceedings of the The Fourth IEEE Symposium on Computers and Communications (ISCC '99), pages 2–8, Washington, DC, USA, 1999. IEEE Computer Society.

[52] W. Du, J. Deng, Y. S. Han, and P. K. Varshney. A pairwise key predistribution scheme for wireless sensor networks. In Proceedings of the 10th ACM conference on Computer and communications security (CCS '03), pages 42–51, New York, NY, USA, 2003. ACM.

[53] W. Du, J. Deng, Y.S. Han, and P. K. Varshney. A key management scheme for wireless sensor networks using deployment knowledge. In Proceeding of IEEE INFOCOM'04, mars 2004.

[54] R. Duncan. An overview of different authentication methods and protocols. In Information Security Reading Room. SANS Institute, octobre 2003.

[55] Eclipse : ite officiel. http ://www.eclipse.org/.

[56] W. El-Hajj, D. Kountanis, A. Al-Fuqaha, and H. Harbi. Optimal hierarchical energy efficient design for MANETs. In Proceeding of the 2006 international conference on Communications and mobile computing (IWCMC'06), pages 287–292, New York, NY, USA, 2006. ACM Press.

[57] A. Abou el kalam. Modèles et politiques de sécurité pour les domaines de la santé et des affaires sociales. PhD thesis, Institut National Polytechnique de Toulouse, 2003. http ://www-verimag.imag.fr/ lakhnech/MODESTE/TheseAnasDiffusable.pdf.

[58] M. Elhdhili, L. Ben Azzouz, and F. Kamoun. A totally distributed cluster based key management model for ad hoc networks. In Third Annual Mediterranean Ad Hoc Networking Workshop, tunisie, juin 2004.

[59] L. Eschenauer and V. Gligor. A key management scheme for distributed sensor networks. In Proceedings of the 9th ACM conference on Computer and communications security (CCS'02), pages 41–47, New York, NY, USA, 2002. ACM.

[60] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next century challenges : Scalable coordination in sensor networks. In Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking (MobiCom '99), pages 263–270, New York, NY, USA, 1999. ACM.

[61] D.F. Ferraiolo and D.R. Kuhn. Role based access control. In Proceedings of the 15th Annual Conference on National Computer Security. National Institute of Standards and Technology, Gaithersburg, pages 554–563, New York, NY, USA, 1992. ACM.

[62] Amos Fiat and Moni Naor. Broadcast encryption. In Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO'93), pages 480–491, London, UK, 1994. Springer-Verlag.

[63] William C. Fifer and Frederick J. Bruno. The low-cost packet radio. In Proceedings of the IEEE, pages 33–42, janvier 1987.

[64] M. Fitzi, J. Garay, S. Gollakota, C. Pandu Rangan, and K. Srinathan. Round-optimal and efficient verifiable secret sharing. In Proceeding of the 2006 Theorical Cryptography Conference, pages 329–342, Berlin, Heidelberg, 2006. Springer-Verlag.

[65] K. Fodor, D. Krupp, G. Biczók, J. L. Gerevich, and K. Sugár. Samson : Smart address management in self-organizing networks. In Proceeding of EUNICE 2005 : Networks and Applications Towards a Ubiquitously Connected World, Boston, MA,USA, 2005. Springer-Verlag.

[66] K. Fokine. Key management in Ad Hoc Networks. PhD thesis, Linkoping University, Suisse, septembre 2002. http ://www.diva-portal.org/diva/getDocument ?urn_nbn_se_liu_diva-1351-1__fulltext.pdf.

[67] Y. Frankel, P. Gemmel, P. MacKenzie, and M. Yung. Optimal-resilience proactive public-key cryptosystems. In Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS '97), pages 384–393, Washington, DC, USA, 1997. IEEE Computer Society.

[68] Matthew S. Gast. 802.11 Wireless Networks : The Definitive Guide. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2002.

[69] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. J. Cryptol., 20(1) :51–83, 2007.

[70] S. Giordano and M. Hamdi. Mobility management : The virtual home region. Technical Report SSC/1999/037, EPFL, Lausanne, Switzerland, octobre 1999. http ://infoscience.epfl.ch/record/52328/files/IC_TECH_REPORT_199937.pdf.

[71] M. Girault. Self-certified public keys. In D.W. Davies, editor, Advances in Cryptology (EUROCRYPT'91. LNCS 547, Springer-Verlag, 1991.

[72] S. Gokhale and P. Dasgupta. Distributed authentication for peer-to-peer networks. In Proceedings of the 2003 Symposium on Applications and the Internet Workshops (SAINT'03 Workshops), pages 347–353, Washington, DC, USA, 2003. IEEE Computer Society.

[73] S. Goldwasser and M.Bellare. Lecture urls on cryptography, août 2001. http ://www-cse.ucsd.edu/ mihir/papers/gb.pdf.

[74] J. Grand. Authentication tokens : Balancing the security risks with business, 2001. http ://malpaso.ru/securid/rr2001-04.pdf.

[75] N. Haller, C. Metz, P. Nesser, and M. Straw. A one-time password system, février 1998. http ://www.faqs.org/rfcs/rfc2289.html.

[76] H. Harney and C. Muckenhim. Group key management protocol (GKMP) architecture, juillet 1997. http ://www.codes-sources.com/rfc.aspx ?rfc=2094.

[77] H. Harney and C. Muckenhim. Group key management protocol (GKMP) specification, juillet 1997. http ://www.faqs.org/rfcs/rfc2093.html.

[78] K. Hatzis, G. Pentaris, P. Spirakis, V. Tampakas, and R. Tan. Fundamental control algorithms in mobile networks. In Proceedings of the 11th Annual ACM Symposium on Parallel Algorithms and Architectures, pages 251–260, 1999.

[79] Olivier Heen, Jean-Pierre Andreaux, and Nicolas Prigent. Improving secure device insertion in home ad-hoc networks. In Y. Deswarte, F. Cuppens, S. Jajodia, , and L. Wand, editors, Security and Protection in Information Processing Systems (IFIP WCC 2004), pages 381–394. Kluwer Academic Publishers, 2004.

[80] C. Hill. The risk of masquerade arising from the storage of biometrics. PhD thesis, Australian National University, 2001. http ://-chris.fornax.net/download/thesis/thesis.pdf.

[81] D.C. Hitchcock. EVALUATION AND COMBINATION OF BIOMETRIC AUTHENTICATION SYSTEMS. PhD thesis, University of Florida, 2003. http ://-purl.fcla.edu/fcla/etd/UFE0002662.

[82] Y.C. Hu, A. Perrig, and D.B. Johnson. Ariadne : A secure on-demand routing protocol for adh̃oc networks. Wireless Networking, 11(1-2) :21–38, 2005.

[83] J.P. Hubaux, L. Buttyan, and S. Capkun. The quest for security in mobile ad hoc networks. In Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing (MobiHoc '01), pages 146–155, New York, NY, USA, 2001. ACM.

[84] I. Ingemarsson, D.T. Tang, and C.K. Wong. A conference key distribution system. In IEEE Transactions on information Theory, volume IT-28, pages 714–720, septembre 1982.

[85] A. Jardosh, E. Belding-Royer, K. Almeroth, and S. Suri. Towards realistic mobility models for mobile ad hoc networks. In Proceedings of the 9th annual international conference on Mobile computing and networking (MobiCom'03), pages 217–229, New York, NY, USA, 2003. ACM Press.

[86] A. Jayasuriya, S. Perreau, A. Dadej, and S. Gordon. Hidden vs. exposed terminal problem in ad hoc networks. In Proceeding of Australian Telecommunication Networks and Applications Conference (ATNAC 2004), pages 52–59, décembre 2004.

[87] JCOP Tools 3.0 (Eclipse Plugin). http ://www.zurich.ibm.com/csc/infosec/jcop_-tools/entry.html.

[88] D. Johnson, D. Maltz, and Y. Hu. The dynamic source routing (DSR) protocol for mobile ad hoc networks. In Internet Engineering Task Force, Internet Draft, avril 2003.

[89] M. Joye and S.M. Yen. Id-based secret-key cryptography. SIGOPS Oper. Syst. Rev., 32(4) :33–39, 1998.

[90] J. Jubin and J.D. Tornow. The DARPA packet radio network protocols. In Proceedings of the IEEE, pages 21–32, 1987.

[91] F. Kargl, S. Schlott, and M. Weber. Identification in ad hoc networks. In Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06), page 233.3, Washington, DC, USA, 2006. IEEE Computer Society.

[92] A. Kerckhoffs. La cryptographie militaire. Journal des sciences militaires, IX :5–38, janvier 1883.

[93] A. Khalili, J. Katz, and W. Arbaugh. Toward secure key distribution in truly ad-hoc networks. In Proceedings of the 2003 Symposium on Applications and the Internet Workshops (SAINT'03 Workshops), pages 342–346, Washington, DC, USA, 2003. IEEE Computer Society.

[94] H. Kikuchi, K. Abe, and S. Nakanishi. Performance evaluation of public-key certificate revocation using k-valued hash tree. In Proceeding of Second International Workshop on Information Security (ISW'99), pages 103–117. Springer-Verlag, 1999.

[95] Y. Kim, D. Mazzochi, and G. Tsudik. Admission control in peer groups. In Proceedings of the Second IEEE International Symposium on Network Computing and Applications (NCA '03), page 131, Washington, DC, USA, 2003. IEEE Computer Society.

[96] J. Kong, H. Luo, K. Xu, D.L. Gu, M. Gerla, and S. Lu. Adaptive security for multi-level ad-hoc networks. Journal of Wireless Communication and Mobile Computing (WCMC), 2 :533–547, 2002.

[97] J. Kong, P. Zerfos, H. Luo, S. Lu, and L. Zhang. Providing robust and ubiquitous security support for mobile ad-hoc networks. In Proceedings of the Ninth International Conference on Network Protocols (ICNP'01), pages 251–260, Washington, DC, USA, 2001. IEEE Computer Society.

[98] B. Leiner, R. Ruth, and A. R. Sastry. Goals and challenges of the DARPA GloMo program. In Proceedings of the IEEE Personal Communications, volume 3, pages 34–43, decembre 1996.

[99] X.S. Li, Y.R. Yang, M.G. Gouda, and S.S. Lam. Batch rekeying for secure group communications. In WWW '01 : Proceedings of the 10th international conference on World Wide Web, pages 525–534, New York, NY, USA, 2001. ACM Press.

[100] I. Litovsky, Y. Métivier, and E. Sopena. Different local controls for graph relabelling systems. Mathematical System Theory, 28 :41–65, 1995.

[101] I. Litovsky, Y. Métivier, and E. Sopena. Graph relabelling systems and distributed algorithms, pages 1–56. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1999.

[102] H. Luo, J. Kong, P. Zerfos, S. Lu, and L. Zhang. Self-securing ad hoc wireless networks. In Proceedings of the Seventh International Symposium on Computers and Communications (ISCC'02), pages 567–574, Washington, DC, USA, 2002. IEEE Computer Society.

[103] H. Luo and S. Lu. Ubiquitous and robust authentication services for ad hoc wireless networks. Technical Report TR-200030, Dept. of Computer Science, UCLA, 2000. http ://camars.kaist.ac.kr/ hyoon/courses/cs710_2002_fall/2002cas/security/papers/

[104] O. Chae M. Mizanur Rahman, M. Abdullah-Al-Wadud. Performance analysis of leader election algorithms in mobile ad hoc networks. IJCSNS International Journal of Computer Science and Network Security, 8(2) :257–263, février 2008.

[105] N. Malpani, J. Welch, and N. Vaidya. Leader election in mobile networks. In Proceeding of the fourth International Workshop on Discrete Algorithms and Methods for Mobile Computing and communications, Boston, MA, août 2000.

[106] T. Matsumoto. Gummy and Conductive Silicone Rubber Fingers, pages 574–576. Springer-Verlag, London, UK, 2002.

[107] V. Matyas and Z. Riha. Biometric authentication systems. Technical report, ECOM-MONITOR, 2000. http ://www.fi.muni.cz/reports/files/older/FIMU-RS-2000-08.pdf.

[108] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. Handbook of Applied Cryptography. CRC Press, Inc., Boca Raton, FL, USA, 1996.

[109] Y. Métivier and E. Sopena. Graph relabelling systems : a general overview. Computers and artificial intelligence, 16 :167–185, 1997.

[110] S. Mitra. Iolus : A framework for scalable secure multicasting. In proceedings of ACM SIGCOMM'97 Conference on Applications, Technologies, Architectures and Procols for Computer Communication, volume 27, pages 277–288, New York, NY, USA, 1997. ACM.

[111] R. Molva and A. Pannetrat. Scalable multicast security in dynamic groups. In CCS '99 : Proceedings of the 6th ACM conference on Computer and communications security, pages 101–112, New York, NY, USA, 1999. ACM Press.

[112] Muscle, mouvement for the use of smart card in a linux environment. http ://www.linuxnet.com/.

[113] M. Musolesi, S. Hailes, and C. Mascolo. An ad hoc mobility model founded on social network theory. In Proceedings of the 7th ACM international symposium

on Modeling, analysis and simulation of wireless and mobile systems (MSWiM'04), pages 20–24, New York, NY, USA, 2004. ACM Press.

[114] M. Naor and K. Nissim. Certificate revocation and certificate update. In Proceedings of the 7th conference on USENIX Security Symposium (SSYM'98), pages 17–30, Berkeley, CA, USA, 1998. USENIX Association.

[115] M. Narasimba, G. Tsudik, and J.H. Yi. On the utility of distributed cryptography in p2p and MAnets : the case of membership control. In Proceedings of the 11th IEEE International Conference on Network Protocols (ICNP'03), pages 336–345, Washington, DC, USA, 2003. IEEE Computer Society.

[116] N.Asokan and P. Ginzboorg. Key agreement in ad-hoc networks. Computer Communications, 23(17) :1627–1637, 2000.

[117] .NET compact framework - .NET CF. http ://www.asp-php.net/tutorial/asp.net/net-compact-framework.php.

[118] J.L. Mu noz, J. Forne, O. Esparza, and M. Soriano. Certificate revocation system implementation based on the merkle hash tree. Int. J. Inf. Secur., 2(2) :110–124, 2004.

[119] NS2On-line Manual. http ://www.isi.edu/nsnam/ns/ns-documentation.html.

[120] R. Ogier, F. Templin, and M. Lewis. Topology dissemination based on reverse-path forwarding (TBRPF), 2004. http ://www.rfc-archive.org/getrfc.php?rfc=3684.

[121] Ps/sc homepage. http ://www.gemplus.com/techno/pcsc/.

[122] T.P. Pedersen. A threshold cryptosystem without a trusted party. In D. Davies, editor, EUROCRYPT'91, pages 526–552. LNCS, 1991.

[123] Adrian Perrig, Dawn Song, and J. D. Tygar. Elk, a new protocol for efficient large-group key distribution. In Proceedings of the 2001 IEEE Symposium on Security and Privacy (SP '01), pages 247–262, Washington, DC, USA, 2001. IEEE Computer Society.

[124] R. Poovendran, S. Corson, and J. Baras. A shared key generation procedure using fractional keys. In IEEE Milcom 98, octobre 1998.

[125] J. Powell and T. Camp. Improving location services with prediction. In ICWN 2006, pages 153–159, 2006.

[126] Nicolas Prigent, Christophe Bidan, Jean-Pierre Andreaux, and Olivier Heen. Secure long term communities in ad hoc networks. In Proceedings of the 1st ACM workshop on Security of ad hoc and sensor networks (SASN '03), pages 115–124, New York, NY, USA, 2003. ACM.

[127] L.G. Roberts. ALOHA packet system with and without slots and capture. SIGCOMM Comput. Commun. Rev., 5(2) :28–42, 1975.

[128] S. Rührup. Position-based Routing Strategies. PhD thesis, Université de Paderborn, Allemagne, 2006. http ://wwwcs.uni-paderborn.de/cs/ag-madh/WWW/sr/thesis.pdf.

[129] R. Sandhu and P. Samarati. Authentication, access control, and audit. ACM Comput. Surv., 28(1) :241–243, 1996.

[130] Damien Sauveron. Etude et réalisation d'un environnement d'expérimentation et de modélisation pour la technologie Java Card. Application à la sécurité. PhD thesis, Université Bordeaux I, Ecole Doctorale de Mathématiques et d'Informatique, Talence, France, 2004. http ://damien.sauveron.free.fr/publications/TheseSauveron.pdf.

[131] N. Saxena, G. Tsudik, and J.H. Yi. Access control in ad hoc groups. In blabla, 2000.

[132] N. Saxena, G. Tsudik, and J.H. Yi. Admission control in peer-to-peer : Design and performance evaluation. In Proceedings of the 1st ACM workshop on Security of ad hoc and sensor networks (SASN'03), pages 104–113, New York, NY, USA, 2003. ACM.

[133] N. Saxena, G. Tsudik, and J.H. Yi. Identity-based access control for ad hoc groups. In Proceedings of the 7th International Conference on Information Security and Cryptology (ICISC'04), Berlin, Heidelberg, 2004. Springer-Verlag.

[134] What's so smart about smart cards ?, 2001. http ://www.gemplus.com/smart/cards/-basics/download/smartcardforum.pdf.

[135] Bruce Schneier. Applied cryptography (2nd ed.) : protocols, algorithms, and source code in C. John Wiley & Sons, Inc., New York, NY, USA, 1995.

[136] S. Setia, S. Koussih, S. Jajodia, and E. Harder. Kronos : A scalable group re-keying approach for secure multicast. In Proceedings of the 2000 IEEE Symposium on Security and Privacy (SP'00), pages 215–228, Washington, DC, USA, 2000. IEEE Computer Society.

[137] A. Shamir. How to share a secret. Commun, ACM, pages 612–613, novembre 1979.

[138] Alan T. Sherman and David A. McGrew. Key establishment in large dynamic groups using one-way function trees. IEEE Transaction of Software Engineering, 29(5) :444–458, 2003.

[139] Clay Shields and J. J. Garcia-Luna-Aceves. KHIP - a scalable protocol for secure multicast routing. In Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication (SIGCOMM '99), pages 53–64, New York, NY, USA, 1999. ACM Press.

[140] V. Shoup. Practical threshold signatures. In EUROCRYPT'00, Berlin, Heidelberg, 2000. Springer-Verlag.

[141] P. Smacchia. Pratique de .NET 2 et C# 2. O'Reilly, 2005.

[142] SpringCard Developers Toolkit Reference version 0.2, 2003.

[143] F. Stajano and R. Anderson. The resurrecting duckling : Security issues for adhoc wireless networks. In Proceedings of the 7th International Workshop on Security Protocols, pages 172–194, London, UK, 2000. Springer-Verlag.

[144] M. Steiner. Secure Group Key Agreement. PhD thesis, Universitat des Saarlandes, 2002. http ://www.semper.org/sirene/publ/Stei_02.thesis-final.pdf.

[145] M. Steiner, G. Tsudik, and M. Waidner. Cliques : A new approach to group key management. In Proceedings of the The 18th International Conference on Distributed Computing Systems (ICDCS '98), pages 380–387, Washington, DC, USA, mai 1998. IEEE Computer Society.

[146] F. Tari, A. Ant Ozok, and S. H. Holden. A comparison of perceived and real shoulder-surfing risks between alphanumeric and graphical passwords. In Proceedings of the second symposium on Usable privacy and security SOUPS'06), pages 56–66, New York, NY, USA, 2006. ACM.

[147] A. Tastinger. Privacy Digest : Books : PGP : Pretty Good Privacy. O'Reilly, 1994.

[148] Trusted computing group. http ://www.trustedcomputinggroup.org/.

[149] Y.C. Tseng, C.S. Hsu, and T.Y. Hsieh. Power-saving protocols for IEEE 802.11-based multi-hop ad hoc networks. Comput. Networks, 43(3) :317–337, 2003.

[150] E. P. Vasilakopoulou, G. E. Karastergios, and G. D. Papadopoulos. Design and implementation of the hiperlan/2 protocol. SIGMOBILE Mob. Comput. Commun. Rev., 7(2) :20–32, 2003.

[151] S. Vasudevan, B. DeCleene, J. Kurose, and K. Towsley. Secure leader election in wireless ad hoc networks. Technical Report TR 01-50, UMass Computer Science. 2001.

[152] S. Vasudevan, N. Immerman, J. Kurose, and K. Towsley. A leader election algorithm for ad hoc networks. Technical Report TR 03-01, UMass Computer Science. 2003.

[153] S. Vasudevan, J. Kurose, and K. Towsley. Design and analysis of a leader election algorithm for mobile ad hoc networks. In Proceedings of the Network Protocols, 12th IEEE International Conference (ICNP '04), pages 350–360, Washington, DC, USA, 2004. IEEE Computer Society.

[154] Visual studio 2005 solution center. http ://support.microsoft.com/ph/3041.

[155] D. Wallner, E. Harder, and R. Agee. Key management for multicast : Issues and architectures, juin 1999. ftp ://ftp.rfc-editor.org/in-urls/rfc2627.txt.

[156] P. Wang and S. Yanushkevich. Biometric technologies and applications. In Proceedings of the 25th conference on Proceedings of the 25th IASTED International Multi-Conference (AIAP'07), pages 226–231, Anaheim, CA, USA, 2007. ACTA Press.

[157] C. Wong, M. Gouda, and S. Lam. Secure group communication using key graphs. In Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication, pages 68–79, 1998.

[158] C. Wong, M. Gouda, and S. Lam. Secure group communication using key graphs. In Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication, pages 16–30, 1998.

[159] H. Xiao, W. Seah, A. Lo, and K. Chua. A flexible quality of service model for mobile ad-hoc networks. In Proceedings in the Vehicular Technology Conference (IEEE VTC 2000-Spring), volume 1, Tokyo, Japon, mai 2000.

[160] G. Yang, L.J. Chen, T. Sun, B. Zhou, and M. Gerla. Ad-hoc storage overlay system (asos) : A delay-tolerant approach in MAnets. In Proceedings of the 3rd IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS 2006), pages 296–305, Vancouver, Canada, octobre 2006.

[161] Y.R. Yang, X.S. Li, X.B. Zhang, and S.S. Lam. Reliable group rekeying : a performance analysis. SIGCOMM Comput. Commun. Rev., 31(4) :27–38, 2001.

[162] S. Yi and R. Kravets. MOCA : Mobile certificate authority for wireless ad hoc networks. In Proceedings of the 2nd Annual PKI Research Workshop (PKI 03), 2003.

[163] J. Yoon, B. Noble, M. Liu, and M. Kim. Building realistic mobility models from coarse-grained traces. In Proceedings of the 4th international conference on Mobile systems, applications and services (MobiSys 2006), pages 177–190, New York, NY, USA, 2006. ACM Press.

[164] H. Yu, P. Martin, and H. Hassanein. Cluster-based replication for large-scale mobile ad-hoc networks. In Proceedings of the Internation Conference on Wireless Networks, Communications and Mobile Computing, volume 1, pages 552–557, juin 2005.

[165] M. Guerrero Zapata. Secure ad hoc on-demand distance vector routing. SIGMOBILE Mob. Comput. Commun. Rev., 6(3) :106–107, 2002.

[166] X. Zeng, R. Bagrodia, and M. Gerla. GloMoSim : a library for parallel simulation of large-scale wireless networks. In Proceedings of the twelfth workshop on Parallel and distributed simulation (PADS'98), pages 154–161, Washington, DC, USA, 1998. IEEE Computer Society.

[167] L. Zhou and Z. Haas. Securing ad hoc networks. In IEEE Network, 13(6), novembre/décembre 1999.

Appendices

Annexe A

Démonstration de l'algorithme d'élection de *leader*

A.1 Méthodologie de démonstration d'un système de réécriture

Pour démontrer la validité de notre algorithme, nous nous servons des notions et des outils présentés par I. Litovski, Y. Métivier et E. Sopena en [101].

A.1.1 Définitions

On représente les opérations d'un algorithme par un système de réécriture de graphes. Nous nous trouvons dans le cas d'un graphe G non connexe, composé de plusieurs sous-graphes connexes, aussi dits îlots. Dans chacun des îlots, les sommets sont étiquetés par une fonction λ : V(G) $\rightarrow$ L, avec V(G) l'ensemble des sommets de G et L un alphabet. On note le graphe considéré (G,λ).

Un sous graphe connexe ou îlot est noté (I, ν), avec ν_X le cardinal de $\nu^{-1}(i, X, n_i, P_i)$ comptant le nombre de sommets possédant le paramètre X. Par exemple, si LEAD est un *leader*, $\nu^{-1}(i, LEAD, n_i, P_i)$ est le nombre de nœuds de *leader* LEAD.

Un système de réécriture $\mathcal{R}$ est défini par $\mathcal{R}$ = (L, I, P, $>$) avec :
- L, un ensemble d'étiquettes ;
- I, un ensemble d'étiquettes initiales représentant les étiquettes des nœuds initiaux (ici I = L) ;
- un ensemble de règles P, avec P = A_w (pour les règles d'apparitions/disparitions, w $\in$ [1 ;2]), C_x (pour les règles de connexions, x $\in$ [1 ;13]), M_y (pour les règles de mise à jour, y $\in$ [1 ;11]), D_z (pour les règles liées à une déconnexion, z $\in$ [1 ;6]).

Définition 1. Une relation binaire R sur E est dite nœthérienne si toute partie non vide A de E contient un élément minimal $a_0 \prec A$ i.e. tel que $\forall a \in A$, $a \neq a_0 => a \not\prec a_0$.

Remarque la relation d'ordre $>$ sur un alphabet est une relation binaire nœthérienne.

Définition 2. Soit R une relation binaire sur un ensemble X, R' une relation binaire sur un ensemble X' et γ une application de X dans X'. La relation R' est **compatible** avec la relation R via γ si pour tout a, b de X on a : $aRb => \gamma(a)R'\gamma(b)$

Lemme 1. Soit R une relation binaire sur un ensemble X, R' une relation binaire sur un ensemble X' et γ une application de X dans X' telle que la relation R' est compatible avec la relation R via γ. Si R' est nœthérienne alors R l'est aussi.

Remarque : toute relation compatible avec la relation d'ordre $>$ est nœthérienne.

Définition 3. Un système de réécriture $\mathcal{R}$ est nœthérien s'il n'existe aucune chaîne infinie de $\mathcal{R}$-réécriture (applications de règles de réécriture défini dans P du système $\mathcal{R}$)).

A.1.2 Validité d'un algorithme

Pour prouver la validité d'un algorithme, il faut prouver que l'algorithme se termine et qu'il est correct :

- un algorithme A se termine si, et seulement si, le système de réécriture est nœthérien (voir définition 3).
- un algorithme A est correct si le système de réécriture est correct.

A.1.3 Validité d'un système de réécriture

Il est beaucoup plus facile de démontrer la validité d'un système de réécriture, la méthodologie étant la suivante :

1. il faut d'abord prouver que le système de réécriture est nœthérien (voir définition 3).

2. il faut ensuite déterminer un système de prédicats satisfait par chaque graphe étiqueté initial et qui est préservé par chaque règle de réécriture. Lorsque plus aucune règle de réécriture ne peut s'appliquer, on dit que le graphe est R-irréductible. Si dans le graphe final tous les nœuds sont dans l'état souhaité alors le système de réécriture est correct.

A.2 Preuve de l'algorithme d'élection d'un *leader*

A.2.1 Terminaison

Un algorithme adapté à un cadre mobile peut ne jamais atteindre une configuration finale puisqu'on considère que la topologie du réseau peut changer de manière incontrôlée et inconnue. L'algorithme peut donc ne jamais se terminer et les règles de réécriture s'appliquer indéfiniment. On a donc cherché à prouver que l'algorithme tend vers une configuration stable et l'atteint s'il n'y a pas de changement de la topologie pendant un laps de temps suffisant.

Il y a deux parties dans les règles de réécriture que nous avons définies : une première où les connexions entraînent l'application des règles de mise à jour et une deuxième, des règles de déconnexions.

Pour les règles de connexion et de mises à jour, la preuve de terminaison est la suivante :

- soit $\Phi : G_L \rightarrow N$ une application qui associe la valeur ν_{LEAD} à un îlot (I, ν), ν_{LEAD} étant le cardinal de $\nu^{-1}(i, LEAD, n_i, P_i)$ c'est-à-dire le nombre de sommets étiquetés avec le *leader* $LEAD$, le *leader* gagnant. Chaque application des règles C1, C2, C3, M1, M2, M3 fait croître strictement ν_{LEAD} puisque par ces règles tout nœud i de *leader* $lead_i$, tout $lead_i \neq LEAD$ devient $LEAD$.

- soit $\Psi : G_L \rightarrow N$ une application qui associe la valeur ν_{NMIN} à un îlot (I, ν), ν_{NMIN} étant le cardinal de $\nu^{-1}(i, lead_i, NMIN, P_i)$ c'est-à-dire le nombre de sommets dont $NMIN$ est le nombre de sauts minimum vers le *leader*. Chaque application des règles C4, C6, M4, M5 et M6 fait croître strictement ν_{NMIN} puisque par C4, C6, M5 et M6 n_i devient $n_j + 1$ (n_i et n_j nombre de saut jusqu'au *leader* de respectivement i et j). Par l'application de la règle M4, n_i est égal au nombre de saut le plus petit du voisinage incrémenté de 1.

Conclusion :

- Soit R une des règles de l'application Φ. On a alors pour (I, ν) et $(I', \nu') \in G_L$: (I, ν) R $(I', \nu') => \Psi((I, \nu)) > \Psi((I', \nu')$. Par la définition 2 l'ensemble des règles R et $>$ sont compatibles via Ψ et par le lemme 1, comme $>$ est une relation binaire nœthérienne alors l'ensemble des règles R le sont aussi.
  De la même manière, l'ensemble des règles R2 définie pour Φ sont nœthériennes.

- Les règles de connexion restantes n'entraînent que l'apparition d'une nouvelle arête et l'exécution d'aucune autre règle de réécriture puisqu'il n'y a aucun changement d'état d'aucun nœud.

- Enfin, il n'existe aucune intéraction entre les règles de la fontion Ψ et celles de la fonction Φ donc l'ensemble des règles de connexion et de mise à jour sont nœthériennes.

En ce qui concerne les règles de déconnexion, il en existe une première partie qui va éventuellement réétiqueter un certain nombre de nœuds en -1 (descente dans la branche ayant détectée la rupture) puis une deuxième qui va réétiqueter ces nœuds en 0 ou défini (remontée dans l'arbre), puis une dernière qui va réétiqueter les nœuds nuls en nœuds définis (diffusion de l'information du nouveau chemin). La preuve de terminaison est la suivante :

- soit $\Phi : G_L \rightarrow N$ une application qui associe la valeur ν_{NMIN} à chaque îlot (I, ν) où $NMIN = -1$ et sachant que pour tout nœud i, ν_{NMIN} est le cardinal de $\nu^{-1}(i, lead_i, NMIN, p_i)$. Chaque application de la règle D4 fait strictement croître ν_{NMIN} puisque par cette règle pour tout nœud i de nombre de saut vers le *leader* n_i on obtient $n_i = -1$.

- soit $\Psi : G_L \rightarrow N$ une application qui associe la valeur ν_{NMIN} à chaque îlot (I, ν) où $NMIN \neq -1$ et sachant que pour tout nœud i, ν_{NMIN} est le cardinal de $\nu^{-1}(i, lead_i, NMIN, p_i)$. Chaque application des règles D5 à D10 fait strictement croître ν_{NMIN}. En effet, par D5, D7 et D9 s'il existe un nœud i et son voisin j tel que si $n_i < n_j + 1$ alors n_i devient $n_j + 1$. Par D6, D8 et D10 n_i devient nul.

- soit $\gamma : G_L \rightarrow N$ une application qui associe chaque îlot (I, ν) à ν_{NMIN} où $n \neq -1, 0$ avec pour tout nœud i, ν_{NMIN} est le cardinal de $\nu^{-1}(i, lid_e, NMIN, p_i)$. Chaque application de D11 fait strictement croître ν_{NMIN}. En effet, n_i qui est initialement nul devient $n_j + 1$.

- D1, D2, D2', D2" et D3 qui sont les règles de rupture d'arête ne peuvent pas former de boucles infinies.

Conclusion :
- Ces trois applications sont nœthériennes (même preuve que pour les règles de connexion), il ne peut donc pas y avoir de boucle infinie de réécritures liées aux déconnexions.

Entre les règles de réécritures de déconnexion et les autres, seules deux liens entre des règles de réécriture peuvent se faire :

1. les règles D2', D7 ou D9 peuvent entraîner l'application de la règle M5. Cette dernière est prouvée nœthérienne ;
2. C1 peut entraîner D11 qui est aussi prouvée nœthérienne.

Aucune boucle infinie n'est donc possible donc le système de réécriture est nœthérien.

A.3 Correction

La correction correspond à trois propriétés que l'on va chercher à prouver avec le système des prédicats :
- Propriété 1 : un réseau stable tend toujours vers un seul îlot où tous les nœuds connaissent leur plus court chemin jusqu'à un *leader* unique.

- Propriété 2 : les règles de déconnexion gèrent la détection d'un îlot.

- Propriété 3 : l'ensemble des règles permet la gestion de plusieurs évènements de connexion et/ou de déconnexion concurrents.

Propriété 1.
- (P_1) : soit deux îlots $i1$ et $i2$ de respectivement $N1$ et $N2$ nœuds, de *leader* respectivement $l1$ et $l2$. Soit n le nombre total de nœuds, donc $n = N1 + N2$.
  On cherche a prouver que, après une fusion, $\forall x, n = n_1(x) + n_2(x)$, avec n_l le nombre de nœuds de *leader* $l1$ (gagnant), n_2 le nombre de nœuds de *leader* $l2$ et x le nombre

d'applications des règles considérées (C1 à C3 et M1 à M3). Pour $x = 0$, $n_1(0) = N1$ et $n_2(0) = N2$.

- Préambule Pr1 : seule une connexion entre deux nœuds d'îlots différents permet de se retrouver dans un îlot avec deux *leader*s différents. Or les règles C1, C2 et C3 qui correspondent à ce genre de connexion assurent que celles-ci entraînent l'adoption du *leader* gagnant par le nœud de *leader* perdant. En effet dans chacune des règles on a un nœud i et un nœud j avec $l_j < l_i$ en partie gauche de la réécriture qui donne en partie droite $l_i = l_j$.

- Préambule Pr2 : les règles de réécriture M1 à M3 peuvent alors s'appliquer puisqu'elles concernent deux nœuds déjà liés et de *leaders* différents. Aucune autre règle ne peut s'appliquer à cette configuration.

- pour $x = 1$ c'est-à-dire lors d'une connexion entre les deux îlots (application de la règle C1 ou C2) on a :
  $n_1(1) = n_1(0) + 1 = N1 + 1$ et $n_2(1) = n_2(0) - 1 = N2 - 1$.
  En effet, dans chacune des règles M1 à M3 on a un nœud i et un nœud j avec $l_j < l_i$ en partie gauche de la réécriture qui donne en partie droite $l_i = l_j$.
- en appliquant ensuite les règles M1, M2 et/ou M3 on suppose que l'hypothèse est vraie pour $x = k - 1$. On a : $n_1(k-1) = N1 + k - 1$ et $n_2(k) = N2 - k + 1$.
- par l'application d'une des règles M1, M2 ou M3 (x = k) entre un nœud i de *leader* $l1$ et un nœud j de *leader* $l2$, le nœud j devient de *leader* $l2$. On a donc :
  $n_1(k) = n_1(k-1) + 1 = N1 + k - 1 + 1 = N1 + k$ et $n_2(k) = n_2(k-1) - 1 = N2 - k + 1 - 1 = N2 - k$.

On a prouvé par récurrence que $\forall x, n_1(x) = N1 + x$ et $n_2 = N2 - x$. On a bien que $\forall x, n = n_1(x) + n_2(x) = N1 + x + N2 - x = N1 + N2$. Enfin, si $x = N2$ alors $n = n_1(N2)$ car $n_2(N2) = N2 - N2 = 0$.

- (P_2) : soit un îlot dans un état stable (un unique *leader* et tous les nœuds connaissent leur plus court chemin vers le *leader* (nombre de saut minimum et prédécesseur). Trois types d'évènements modifient le voisinage et nécessitent une mise à jour des chemins.

  1. une fusion entre deux îlots nécessite la mise à jour des chemins des nœuds de l'îlot perdant. Avec la même preuve par récurrence que dans P_1, on prouve que $n = c_1(x) + c_2(x)$ avec c_1 et c_2 le nombre de nœuds connaissant leur plus court chemin jusqu'à leur *leader* (resp. $l1$ et $l2$). On en conclut que $n = c1(N2)$.

  2. des connexions entre les nœuds d'un même îlot par l'application de la règle C4 peut entraîner un état où tous les nœuds ne connaissent pas le plus petit chemin jusqu'au *leader*. Soit dans l'îlot i, c_o et c_n respectivement les nœuds connaissant leur plus court chemin et ceux ne le connaissant pas. On veut prouver que $\forall x, n = c_o(x) + c_n(x)$ avec $c_o(0) = Co$ et $c_n(0) = Cn$.
     - pour $x = 1$ on a $c_o(1) = c_o(0) + 1 = Co + 1$ et $c_n(1) = c_n(0) - 1 = Cn - 1$.
       En effet, par l'application de chacune des règles M4 et M5 on a un nœud i et un nœud j avec $n_j + 1 < n_i$ en partie gauche de la réécriture qui donne en partie droite $n_i = n_j + 1$.

- en appliquant k - 1 fois les règles M4 ou M5, on suppose que l'on a : $c_o(k-1) = Co + k - 1$ et $c_n(k) = Cn - k + 1$.
- par l'application une nouvelle fois d'une des règles M4 ou M5 ($x = k$) on a : $c_o(k) = c_o(k-1) + 1 = Co + k - 1 + 1 = Co + k$ et $c_n(k) = c_n(k-1) - 1 = Cn - k + 1 - 1 = C2 - k$.

On a prouvé par récurrence que $\forall x, c_o(x) = Co + x$ et $c_n = Cn - x$. On a bien que $\forall x, n = c_o(x) + c_n(x) = Co + x + Cn - x = Co + Cn$. Enfin, si $x = Cn$ alors $n = c_o(Cn)$ car $c_n(Cn) = Cn - Cn = 0$.

3. une rupture de lien entraînant l'application de la règle D2' modifie le chemin d'un nœud i en lui affectant un autre prédécesseur (celui offrant le plus petit chemin). S'en suivent les règles M4 et/ou M5 et la preuve que l'îlot tend vers le bon état est la même que la précédente (connexion entre des nœuds d'un même îlot).
   Remarque : les règles de rupture D2, D2" et D3 modifient le nœud en mettant le chemin à nul, ce que nous ne traitons pas ici. La règle D1 n'entraîne aucun changement.

Propriété 2. Il faut prouver que la déconnexion entraîne la recherche d'un nouveau chemin et que cette recherche est caractérisée par la mise à -1 des chemins des nœuds concernés. Ensuite, il suffit de prouver que ces nœuds en état de recherche retrouvent un nouveau chemin vers le même *leader* ou qu'il y a détection d'une partition le cas échéant.

- (P_3) : soit un îlot i dans un état stable. Seule l'application de la règle D2 entraîne la mise en recherche du nœud ayant détecté la rupture. Il faut que le nœud ait des fils. Remarque : si le nœud n'a pas de fils, il devient son propre *leader* s'il n'a pas de voisin (non frères) proposant un chemin (D3 et D2"), sinon il adopte celui offrant le plus petit chemin (D2'). La règle D1 n'entraîne aucun changement.

- (P_4) un sommet étiqueté -1 ou 0 a un père étiqueté -1 ou 0 ou n'en a pas. L'ensemble des nœuds dans un état non défini sont des nœuds qui sont en recherche d'un nouveau chemin.
  - un nœud ne peut être réétiqueté -1 que par la règle D4 si le prédécesseur est -1 ou 0.
  - un sommet ne peut être réétiqueté 0 que par les règles
    - C2, C3 et M3 lors de la connexion avec un îlot en recherche et où le nœud de l'autre îlot est indéfini ou nul ;
    - la règle D2 provoquée par la rupture du lien avec le père donc le nœud concerné n'a plus de père et met son nombre de saut et son prédécesseur à nul ;
    - la règle D4 provoquée par l'état indéfini ou nul du père. Le nœud concerné ayant des fils il se met à indéfini ;
    - la règle D6 provoquée par l'état indéfini ou nul du père. Le nœud concerné n'ayant pas de fils (autre que des frères) c'est une feuille, il se met à nul ;
    - la règle D8 où le nœud était déjà à indéfini, le nœud se met à nul (tous ces fils sont devenus nul) ;

  Donc les nœuds indéfinis ou nuls ont leur père indéfini ou nul. On remonte jusqu'au

sommet où se trouve un nœud n'ayant pas de prédécesseur, c'est-à-dire celui ayant détecté la rupture. Ainsi on a bien que les nœuds indéfinis ou nuls sont des nœuds qui n'ont plus de chemin vers leur *leader*.

- (P_5) s'il existe un nouveau chemin, tous les nœuds qui étaient à indéfini redeviennent définis du même *leader*.
  - par la règle D5, D7 ou D9 un nœud, dont le prédécesseur est en recherche ou n'existe pas, trouve un chemin défini. La propagation de ce nouveau chemin se fait par l'application de la règle M6. La preuve que tous les nœuds dans l'état indéfini sont remis à jour se fait identiquement à la preuve par récurrence développée en P_2.

- (P_6) : s'il n'existe pas de nouveau chemin, une partition est détectée et un nouveau *leader* est élu.
  - aucun chemin n'a été trouvé donc tous les nœuds sont dans l'état indérfini et les feuilles dans l'état nul (d'après P_4) ;
  - par l'application de la règle D6 ou D8, on prouve par récurrence de la même manière que précédemment que tous les nœuds indéfinis dont tous les fils sont nuls deviennent nul ;
  - par l'application de la règle D10, le nœud ayant détecté la rupture devient *leader* lorsque tous ces fils sont dans l'état nul. Cette règle entraîne la règle D11. On prouve par récurrence (comme précédemment) que l'application de la règle D11 entraîne l'adoption du nouveau *leader* à tous les nœuds de l'îlot.

Propriété 3. Nous devons prouver que plusieurs évènements concurrents peuvent se produire sans poser de problème, i.e. en ne compromettant pas le but d'obtenir des îlot stables. Il existe trois possibilités d'évènements concurrents :

- Dans un même îlot :
  - Connexion - connexion : plusieurs connexions à l'intérieur d'un même îlot peuvent se produire en même temps. Ceci est exprimé par les règles C4 et C5. Le plus court chemin peut alors changer là où les nouvelles liaisons sont apparues. Les règles M4 et M5 assurent ensuite la mise à jour des chemins lorsqu'une modification du chemin d'un voisin a eu lieu.

  - Déconnexion - déconnexion : deux déconnexions sur des arêtes de l'arbre ont lieu dans un même îlot.
    Il s'agit du processus de déconnexion normal dans chacune des branches de l'arbre concernée.

  - Connexion - déconnexion : une déconnexion se produit dans un îlot provoquant la mise en état de recherche des nœuds de la branche concernée.
    - 1er cas : une connexion a lieu entre un des nœuds en état de recherche d'un chemin et un nœud stable.
      Les règles C6 et C7 correspondent à ce cas ; si un des nœuds est nul et l'autre défini (C6) alors le nœud nul adopte le nœud défini comme prédécesseur (le nœud nul a ses fils et petits fils à nul). Si un des nœuds est indéfini et l'autre défini (C7) il ne se passe rien car le nœud indéfini est en recherche et le nœud défini

pourrait très bien être un de ses petits-fils. Dans ces deux cas le nœud défini ne change pas puisque le chemin qui lui est proposé n'est pas un plus court chemin (indéfini ou nul).
La règles C6 entraîne l'application de la règle M6 pour propager le chemin nouvellement trouvé aux fils qui sont forcément nuls eux même.

- 2ème cas : deux des nœuds en recherche d'un chemin se connectent entre eux.
  Ce sont les règles C8 à C13 qui correspondent au cas de la connexion entre les deux nœuds. Il ne se passe rien car aucun des nœuds ne peut offrir à l'autre un chemin vers son *leader*, juste l'ajout d'une arête entre les deux nœuds.

- Dans un îlot différent :
  - Connexion - connexion : ce sont les règles permettant d'avoir un unique *leader* dans un îlot et de conserver un plus court chemin qui sont appliquées normalement à savoir C1, M1 et M2.

  - Déconnexion - déconnexion : deux déconnexions dans des îlots différent on lieu.
    Il s'agit du processus de déconnexion normal dans chacun des îlots concerné.

  - Connexion - déconnexion : une déconnexion se produit sur l'arête de l'arbre considéré provoquant la mise en état de recherche des nœuds de la branche concernée.
    - 1er cas : dans un îlot, une déconnexion se produit provoquant la mise en recherche d'un chemin vers le *leader* des nœuds de la branche concernée. Une connexion a lieu entre un des nœuds en recherche de cet îlot et un nœud défini d'un autre îlot.
      Les règles C1, C2, C3 représentent ce cas : la recherche d'un unique *leader* dans un îlot étant privilégiée, quelque soit l'état du nœud de *leader* gagnant le nœud de *leader* perdant l'adopte. Si le nœud de *leader* gagnant est dans un état indéfini ou nul, le nœud de *leader* perdant se met à nul ; si le nœud de *leader* gagnant a un chemin défini, le nœud de *leader* perdant prend son chemin plus un.
      Une fois la connexion établie l'information est propagée par les règles M1 à M3.

    - 2ème cas : dans deux îlots différents, une déconnexion se produit provoquant la mise en recherche des nœuds de la branche concernée. Une connexion a lieu entre ces deux îlots via deux nœuds chacun en état de recherche.
      Pour ce qui est de la connexion ce sont C2 et C3 qui s'appliquent : encore une fois c'est le *leader* gagnant qui est adopté. Le nœud de *leader* perdant se met directement à nul puisqu'il n'a pas de chemin à offrir avec ses fils. Si l'un d'eux s'est entre temps connecté avec un nœud défini de l'îlot gagnant on retrouve les cas ayant lieu à l'intérieur d'un même îlot.
      La propagation de l'information se fait par la règle M3 qui fait adopter le nouveau *leader* à tous les nœuds fils ou petits-fils du nœud de *leader* perdant et leur attribue l'état nul.

- Tous les cas sont bien traités par les règles de réécriture.

Annexe B

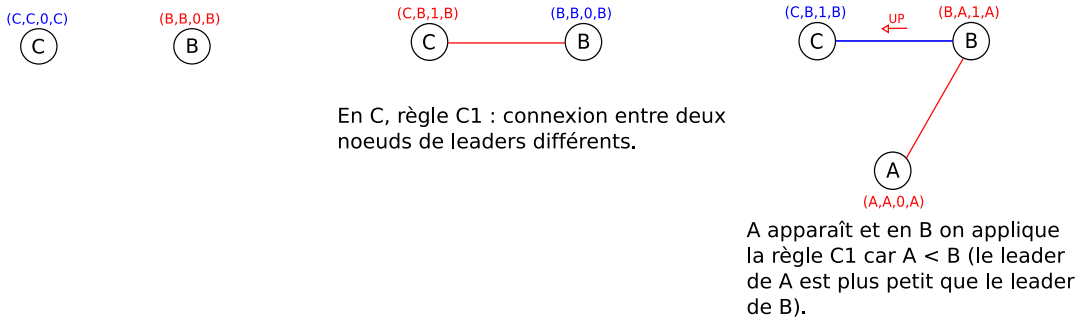
Exemples de déroulements de l'algorithme du *leader*

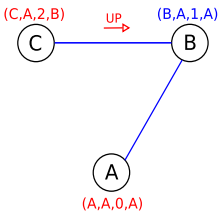
Dans ce chapitre, nous proposons différents exemples de graphes représentant des déroulements de l'algorithme du *leader* décrit au chapitre 4 en fonction des différents évènements pouvant survenir au niveau du réseau.
Lecture des exemples :

- les nœuds sont identifiés par des lettres et la comparaison entre *leaders* se fait sur ces identités selon l'ordre lexicographique.
- l'envoi d'un message par un nœud après modification de son étiquette est représenté par une flèche portant le mot « up » (pour *update*).
- la liste des fils et des voisins d'un nœud avec leur état est maintenue par chaque nœud grâce à un processus régulier de découverte.

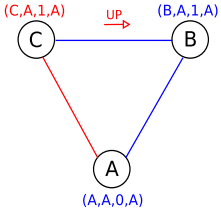
B.1 Connexions

Nous présentons ici des exemples de connexions très simples entre des nœuds qui apparaissent sans appartenir à aucun îlot (ils constituent leur propre îlot).

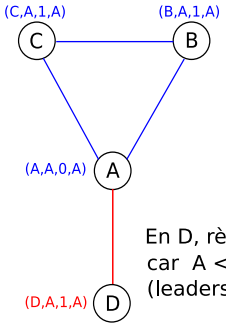




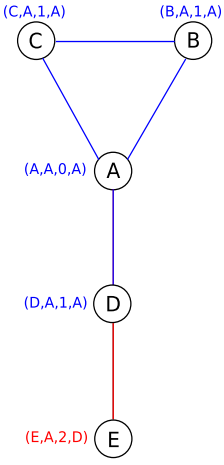
En C, règle M1 : propagation de l'information du leader gagnant.



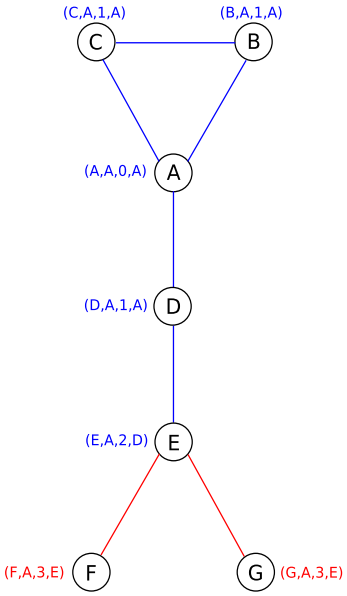
En C, règle C4 : une connexion a lieu à l'intérieur d'un même îlot et le chemin nouvellement proposé est plus court donc on l'adopte.



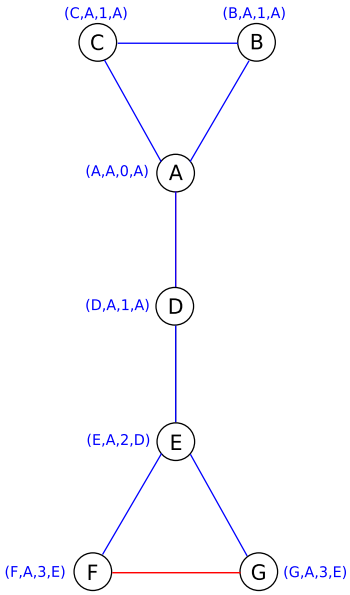
En D, règle C1 car A < D (leaders).



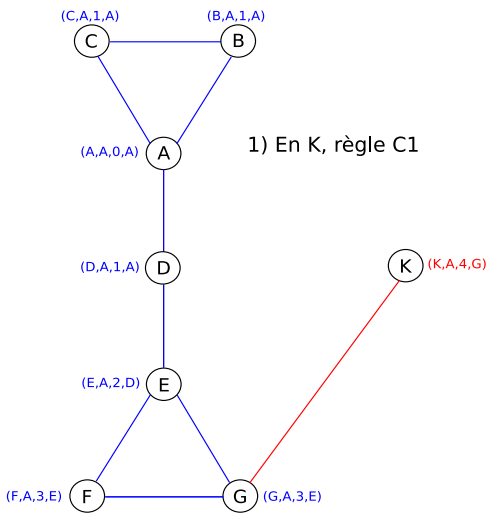
En E, règle C1 car A < E

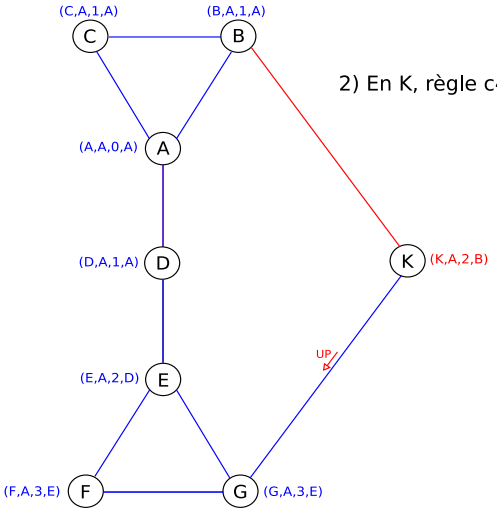


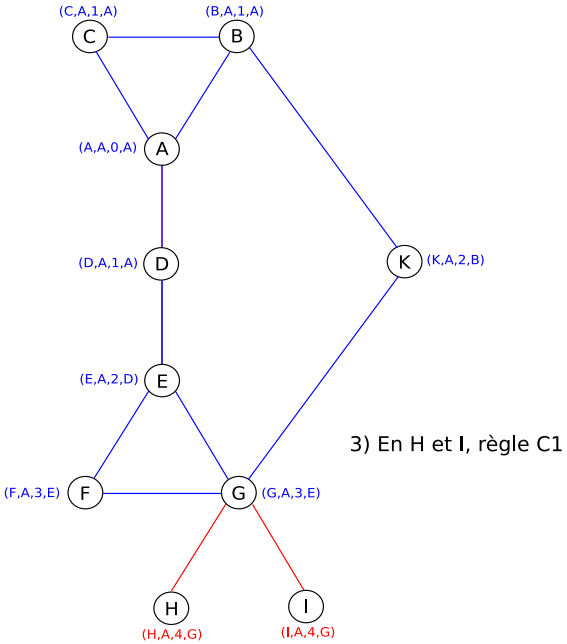
En F et G, règle C1 car F,G < A

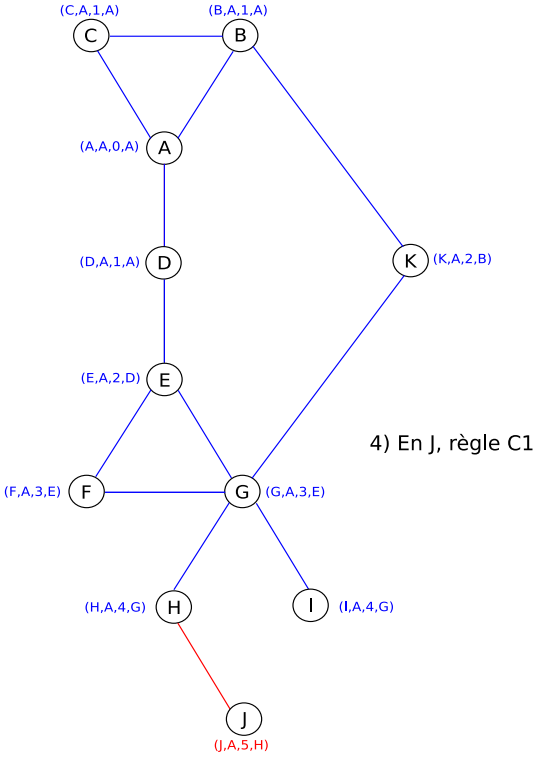


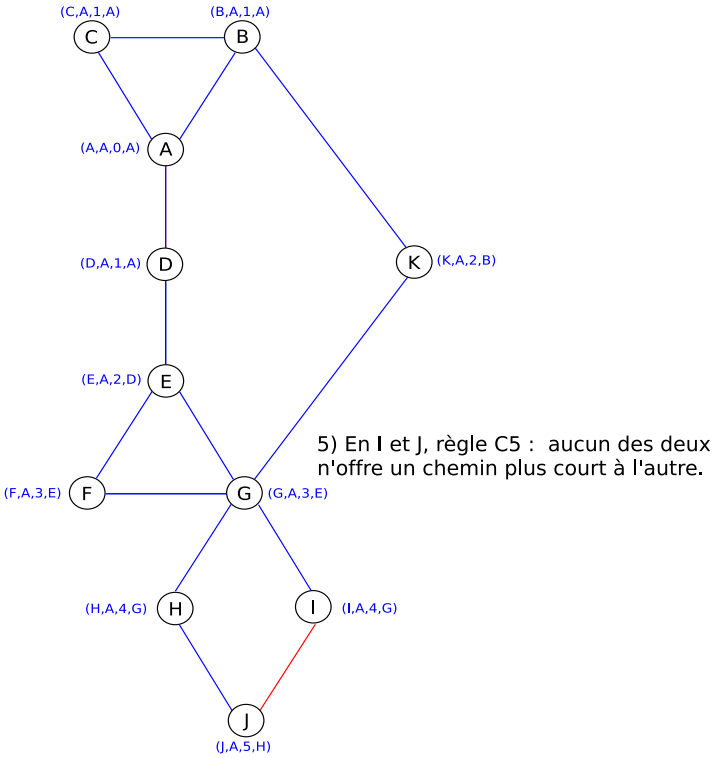
En G, règle C5 : une connexion a lieu à l'intérieur d'un même îlot et le chemin nouvellement proposé n'est pas plus court; G ne change donc pas.


(C,A,1,A)
(B,A,1,A)
C
B
(A,A,0,A)
A
1) En K, règle C1
(D,A,1,A)
D
K
(K,A,4,G)
(E,A,2,D)
E
(F,A,3,E)
F
G
(G,A,3,E)


(C,A,1,A)
(B,A,1,A)
C
B
2) En K, règle c4
(A,A,0,A)
A
(D,A,1,A)
D
K
(K,A,2,B)
UP
(E,A,2,D)
E
(F,A,3,E)
F
G
(G,A,3,E)

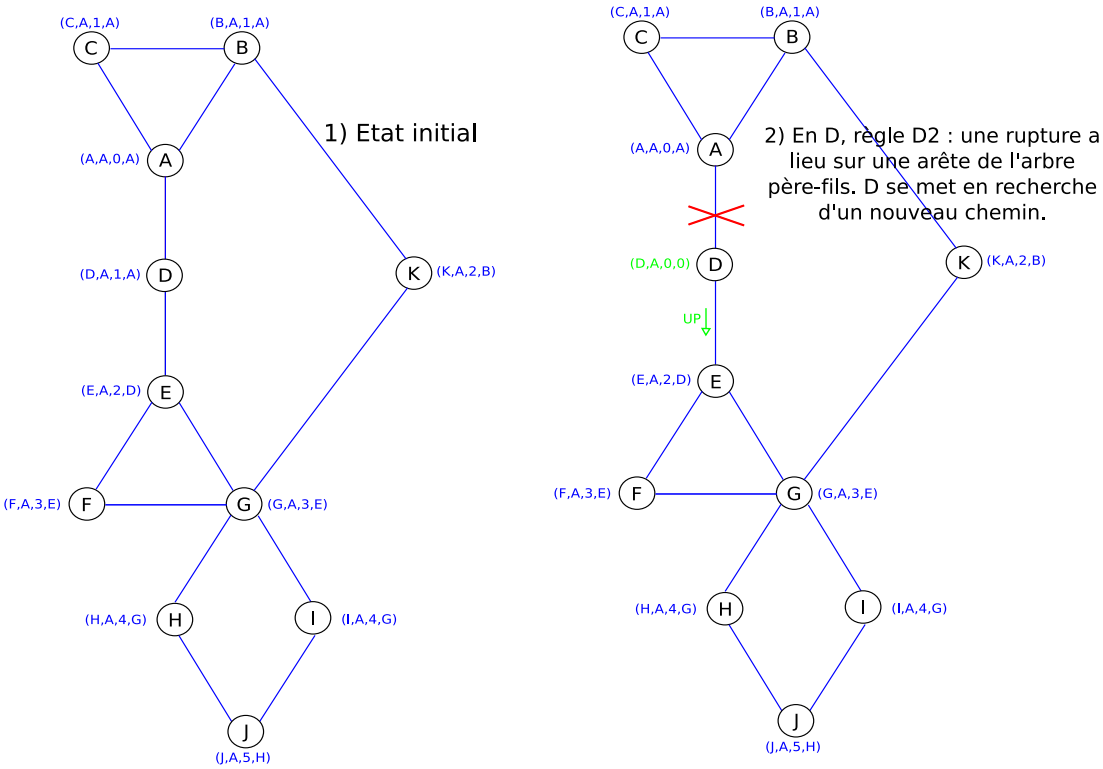

(C,A,1,A)
(B,A,1,A)
C
B
(A,A,0,A)
A
(D,A,1,A)
D
K
(K,A,2,B)
(E,A,2,D)
E
3) En H et I, règle C1
(F,A,3,E)
F
G
(G,A,3,E)
H
(H,A,4,G)
I
(I,A,4,G)

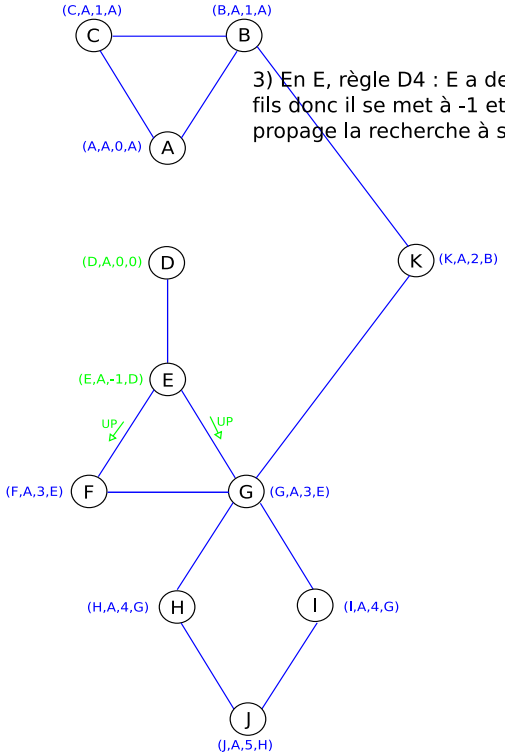

(C,A,1,A)
(B,A,1,A)
C
B
(A,A,0,A)
A
(D,A,1,A)
D
K
(K,A,2,B)
(E,A,2,D)
E
4) En J, règle C1
(F,A,3,E)
F
G
(G,A,3,E)
(H,A,4,G)
H
I
(I,A,4,G)
J
(J,A,5,H)

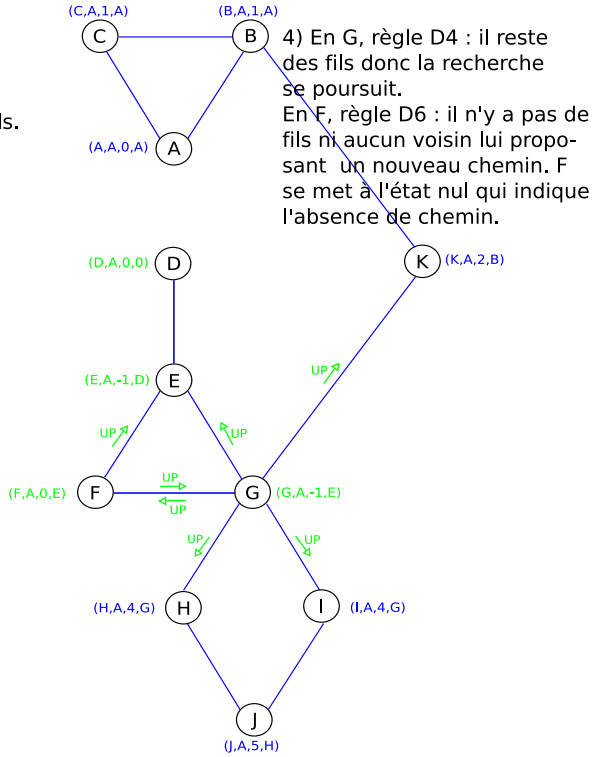

(C,A,1,A)
(B,A,1,A)
C
B
(A,A,0,A)
A
(D,A,1,A)
D
K
(K,A,2,B)
(E,A,2,D)
E
5) En I et J, règle C5 : aucun des deux
n'offre un chemin plus court à l'autre.
(F,A,3,E)
F
G
(G,A,3,E)
(H,A,4,G)
H
I
(I,A,4,G)
J
(J,A,5,H)

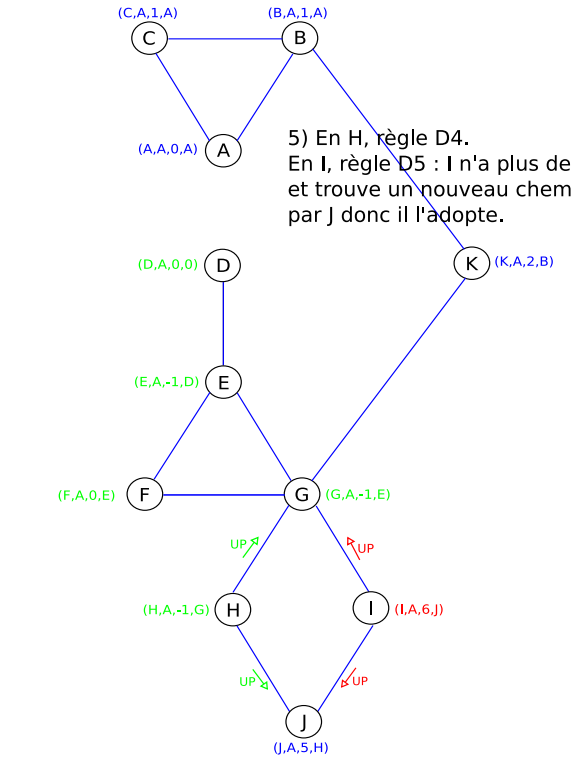
B.2 Déconnexion simple

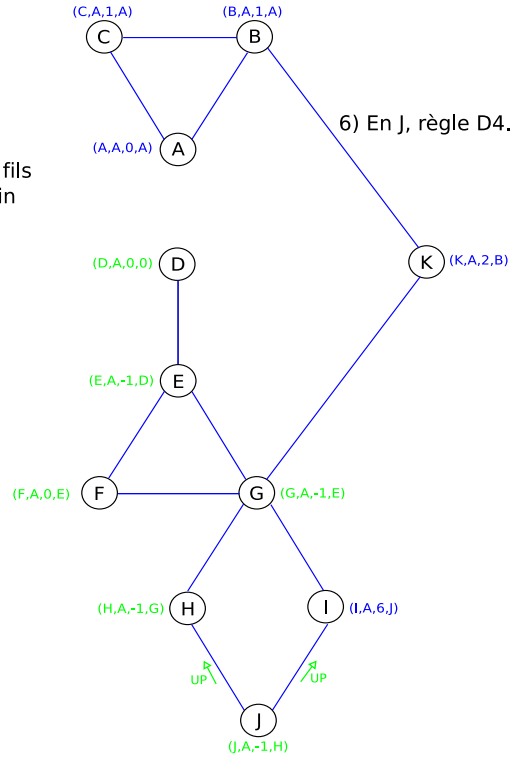
Nous proposons ici l'exemple d'une rupture d'arête dans l'arbre du plus court chemin vers le *leader*, rupture qui entraîne la recherche puis la découverte d'un nouveau chemin.

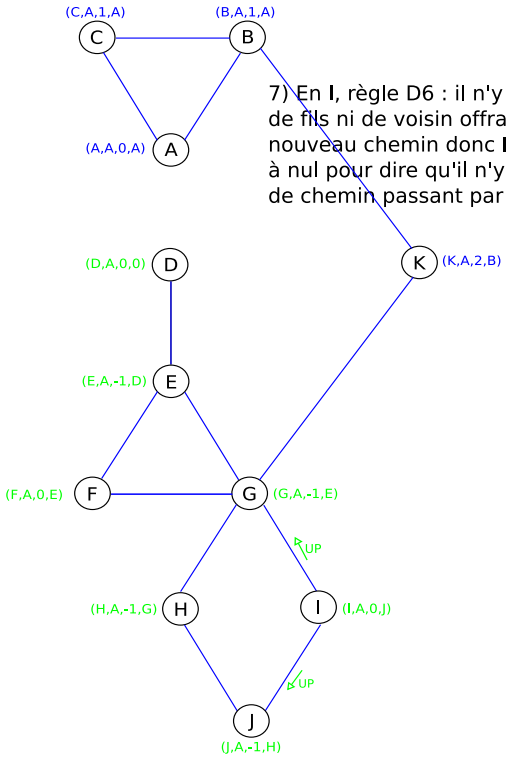


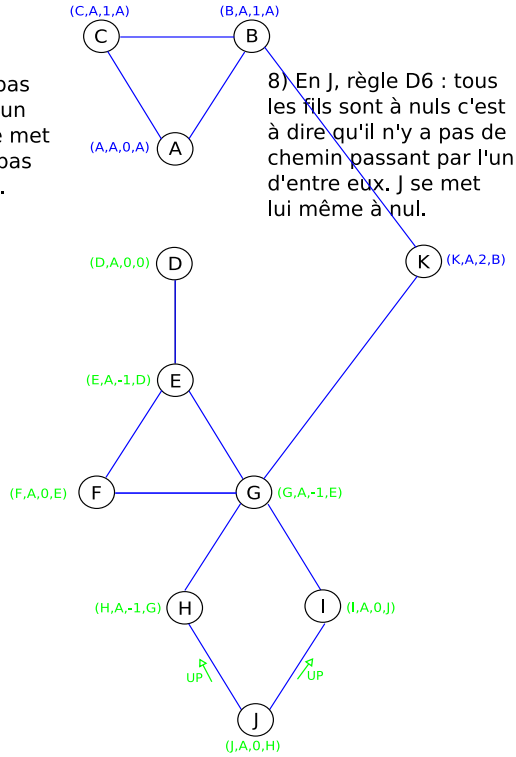

(C,A,1,A)
(B,A,1,A)
C
B
(A,A,0,A)
A
3) En E, règle D4 : E a des fils donc il se met à -1 et propage la recherche à ses fils.
(D,A,0,0)
D
K
(K,A,2,B)
(E,A,-1,D)
E
UP
UP
(F,A,3,E)
F
G
(G,A,3,E)
(H,A,4,G)
H
I
(I,A,4,G)
J
(J,A,5,H)

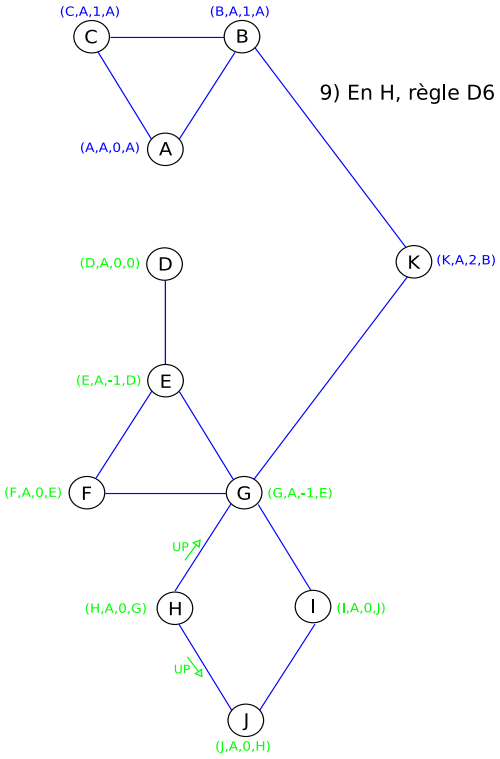

(C,A,1,A)
(B,A,1,A)
C
B
(A,A,0,A)
A
4) En G, règle D4 : il reste des fils donc la recherche se poursuit.
En F, règle D6 : il n'y a pas de fils ni aucun voisin lui proposant un nouveau chemin. F se met à l'état nul qui indique l'absence de chemin.
(D,A,0,0)
D
K
(K,A,2,B)
(E,A,-1,D)
E
UP
UP
UP
UP
(F,A,0,E)
F
UP
UP
G
(G,A,-1,E)
UP
UP
(H,A,4,G)
H
I
(I,A,4,G)
J
(J,A,5,H)

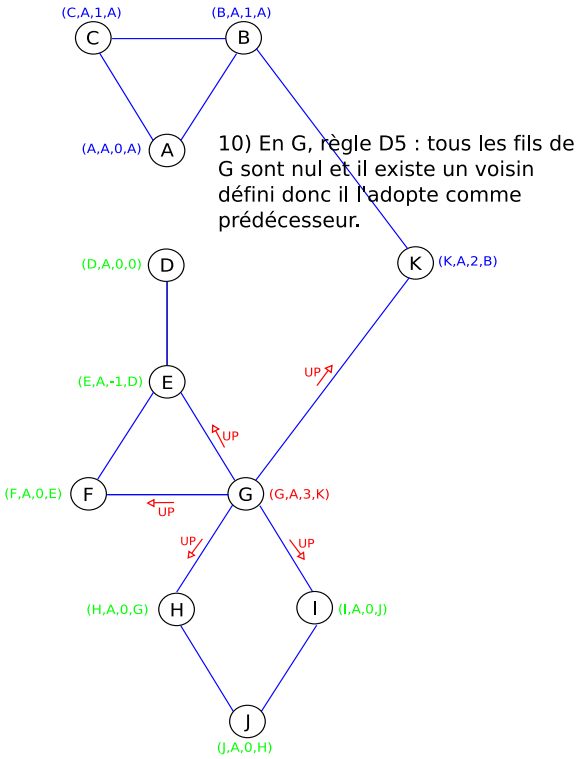

(C,A,1,A)
(B,A,1,A)
C
B
(A,A,0,A)
A
5) En H, règle D4.
En I, règle D5 : I n'a plus de fils et trouve un nouveau chemin par J donc il l'adopte.
(D,A,0,0)
D
K
(K,A,2,B)
(E,A,-1,D)
E
(F,A,0,E)
F
G
(G,A,-1,E)
UP
UP
(H,A,-1,G)
H
I
(I,A,6,J)
UP
UP
J
(J,A,5,H)

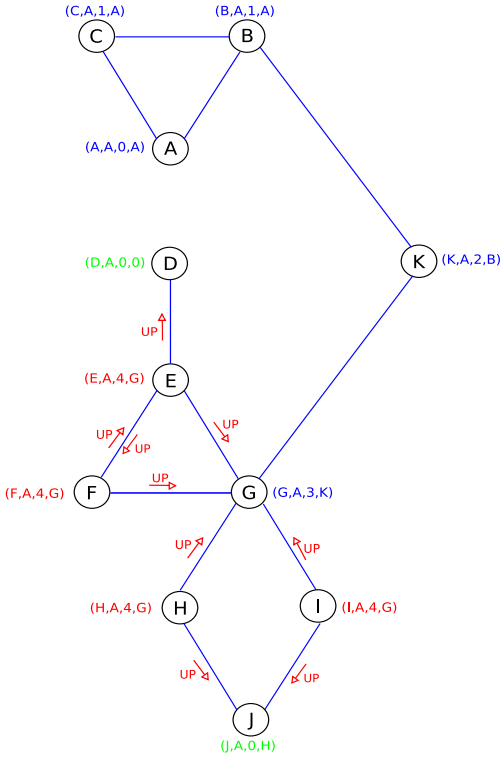

(C,A,1,A)
(B,A,1,A)
C
B
(A,A,0,A)
A
6) En J, règle D4.
(D,A,0,0)
D
K
(K,A,2,B)
(E,A,-1,D)
E
(F,A,0,E)
F
G
(G,A,-1,E)
(H,A,-1,G)
H
I
(I,A,6,J)
UP
UP
J
(J,A,-1,H)


(C,A,1,A)
(B,A,1,A)
C
B
(A,A,0,A)
A
7) En I, règle D6 : il n'y a pas de fils ni de voisin offrant un nouveau chemin donc I se met à nul pour dire qu'il n'y a pas de chemin passant par lui.
(D,A,0,0)
D
K
(K,A,2,B)
(E,A,-1,D)
E
(F,A,0,E)
F
G
(G,A,-1,E)
UP
(H,A,-1,G)
H
I
(I,A,0,J)
UP
J
(J,A,-1,H)

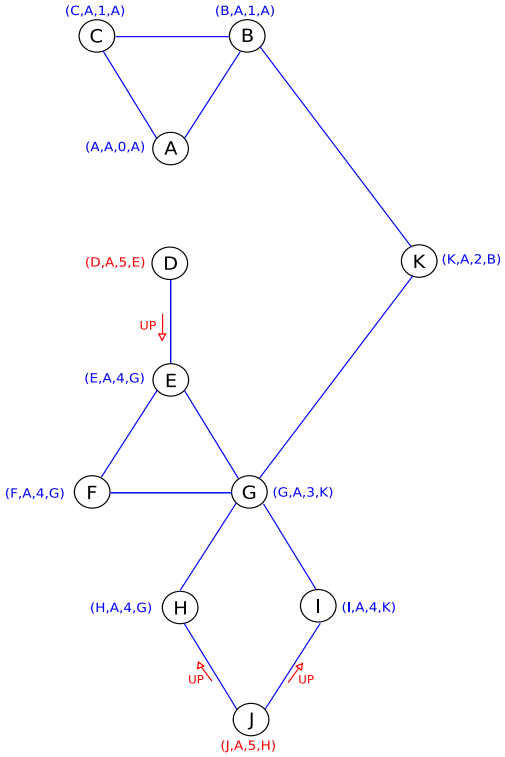

(C,A,1,A)
(B,A,1,A)
C
B
(A,A,0,A)
A
8) En J, règle D6 : tous les fils sont à nuls c'est à dire qu'il n'y a pas de chemin passant par l'un d'entre eux. J se met lui même à nul.
(D,A,0,0)
D
K
(K,A,2,B)
(E,A,-1,D)
E
(F,A,0,E)
F
G
(G,A,-1,E)
(H,A,-1,G)
H
I
(I,A,0,J)
UP
UP
J
(J,A,0,H)


(C,A,1,A)
(B,A,1,A)
C
B
(A,A,0,A)
A
9) En H, règle D6
(D,A,0,0)
D
K
(K,A,2,B)
(E,A,-1,D)
E
(F,A,0,E)
F
G
(G,A,-1,E)
UP
(H,A,0,G)
H
I
(I,A,0,J)
UP
J
(J,A,0,H)


(C,A,1,A)
(B,A,1,A)
C
B
(A,A,0,A)
A
10) En G, règle D5 : tous les fils de G sont nul et il existe un voisin défini donc il l'adopte comme prédécesseur.
(D,A,0,0)
D
K
(K,A,2,B)
(E,A,-1,D)
E
UP
UP
(F,A,0,E)
F
UP
G
(G,A,3,K)
UP
UP
(H,A,0,G)
H
I
(I,A,0,J)
J
(J,A,0,H)



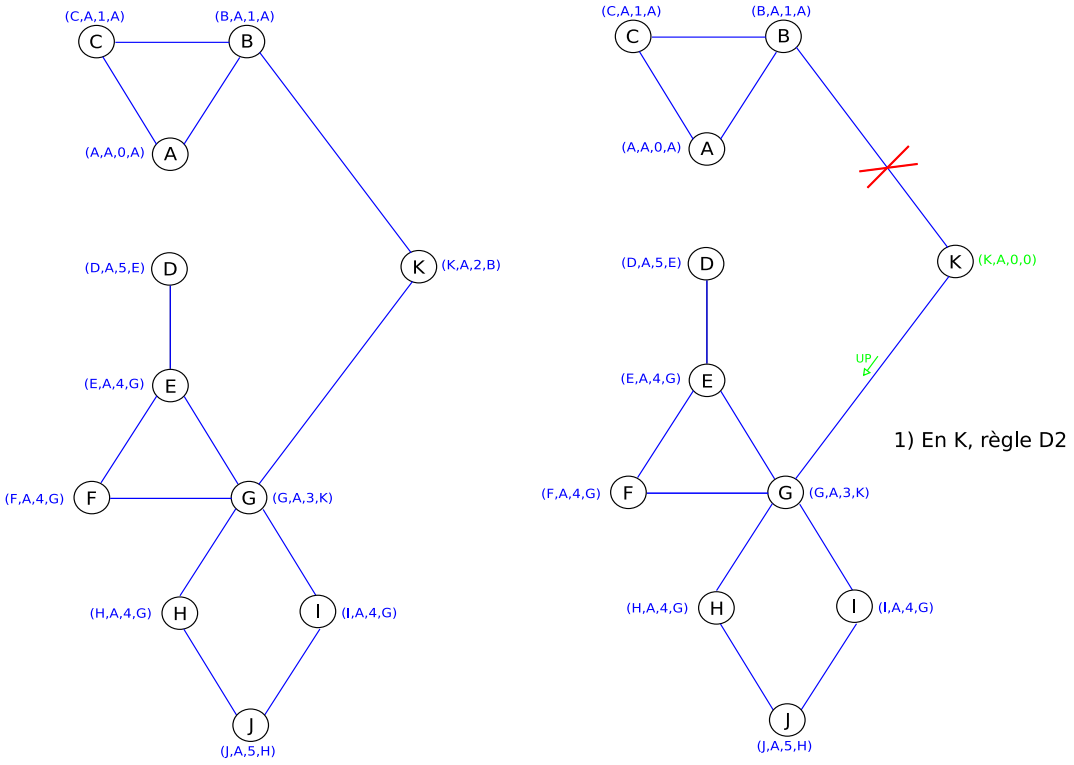
11) En F et **I**, règle M6 : le voisin G de F et de **I** qui était nul devient défini donc F et **I** qui sont toujours à nul l'adoptent comme prédécesseur.
En E, règle D7 : tous les fils de E sont nuls et un chemin existe donc il est adopté.
En H règle D11 : H est à nul et son prédécesseur a adopté un nouveau leader donc il l'adopte.

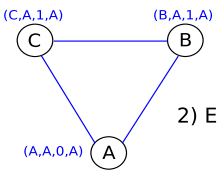


12) En D, règle D9 : le noeud qui a détecté la rupture n'a plus de fils idéfinis (il n'a plus de fils du tout) mais il a un voisin défini, et donc il adopte le chemin que celui-ci lui propose.
En J, règle D11 : J est à nul et son prédécesseur a adopté un nouveau leader donc il l'adopte.

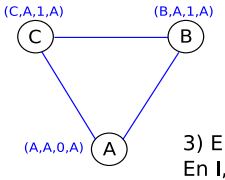
B.3 Détection d'un îlot

Nous détaillons ici le cas d'une partition d'un groupe en deux sous-groupes, suite à la rupture d'une liaison.

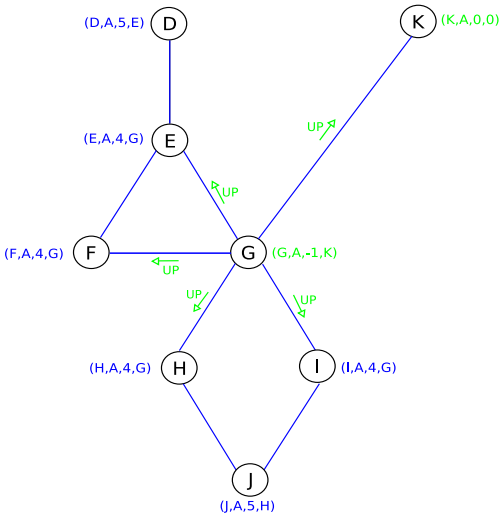


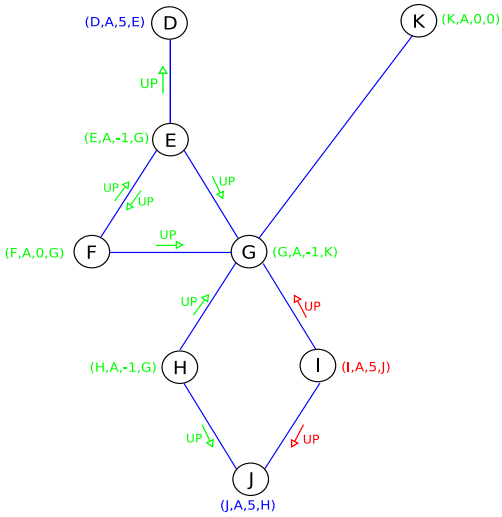


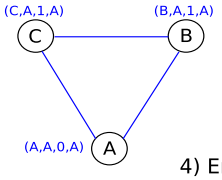
2) En G, règle D4



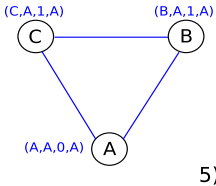
3) En E,F,H règle D4
En **I**, règle D5



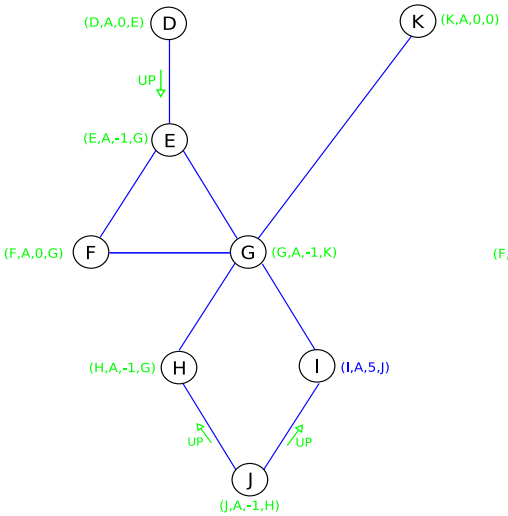


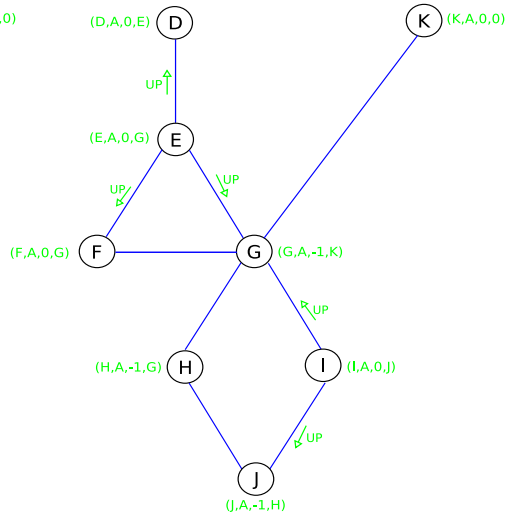


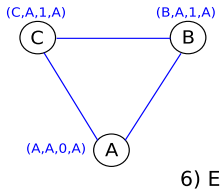
4) En J, règle D4
En D, règle D6



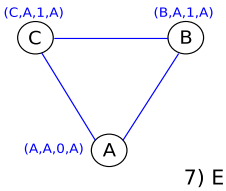
5) En **I**, règle D6
En E, règle D6



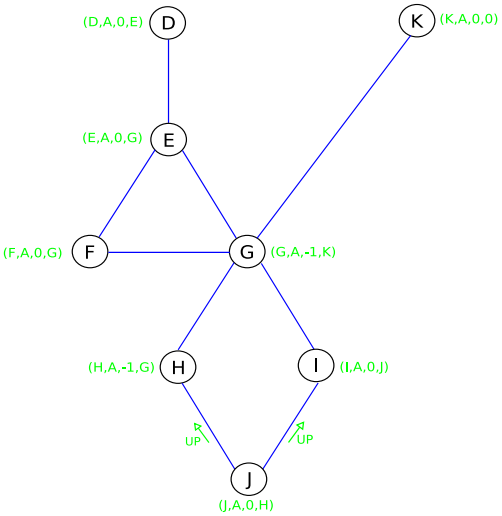


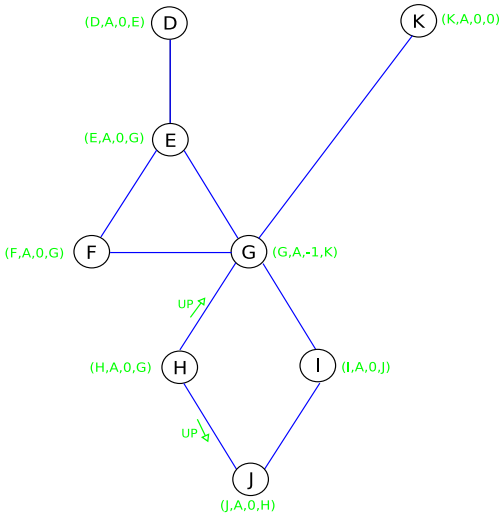


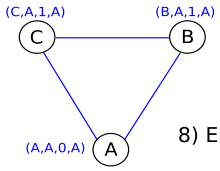
6) En J, règle D6



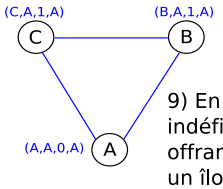
7) En H, règle D6



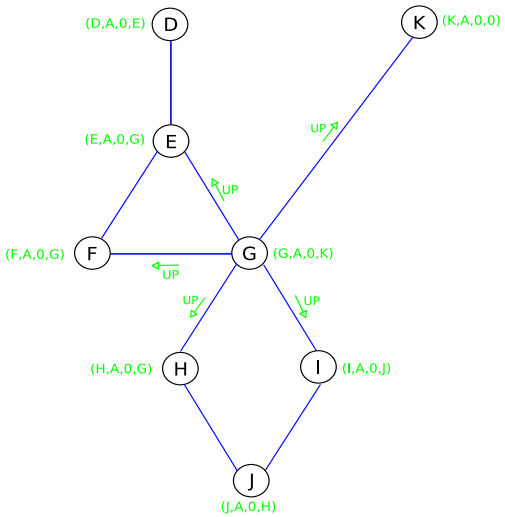


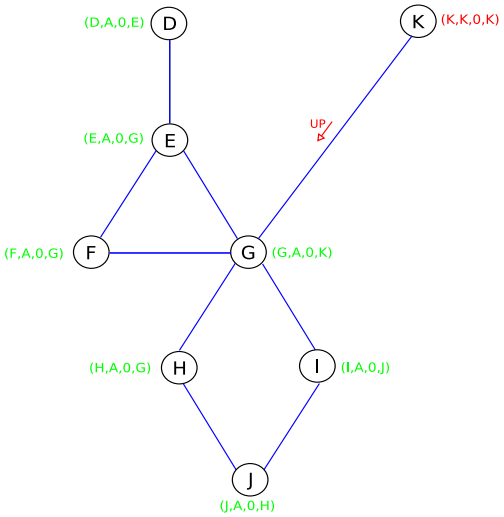


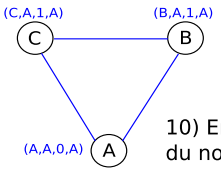
8) En G, règle D6



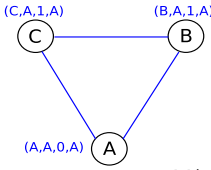
9) En K, règle D10 : K n'a plus de fils indéfini mais il n'a pas de voisin lui offrant un chemin donc il a détecté un îlot et en devient le leader.



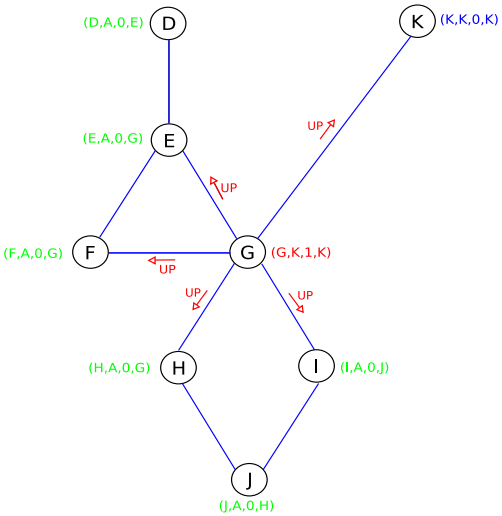


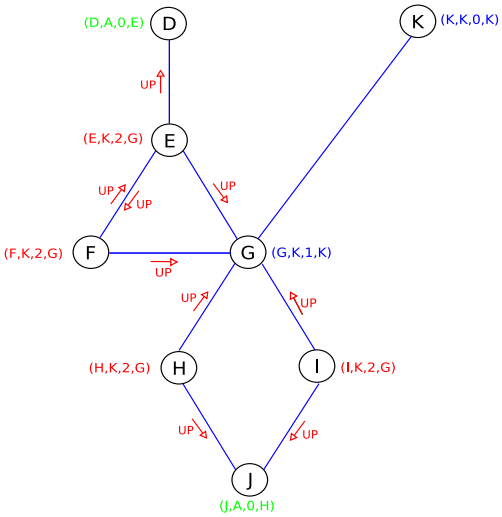


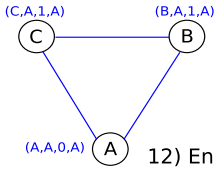
10) En G, règle D11 : mise à jour du nouveau leader



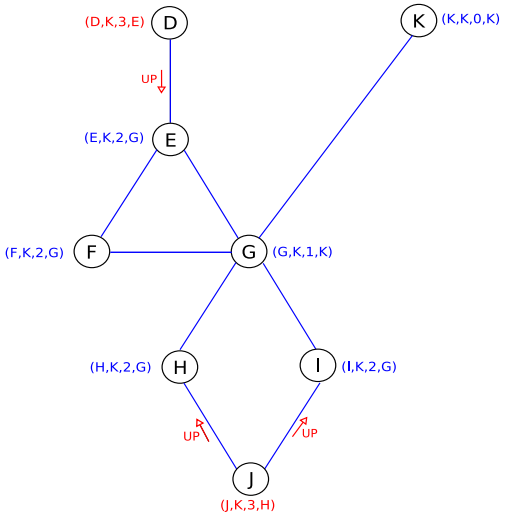
11) En E, F, H, I, règle D11





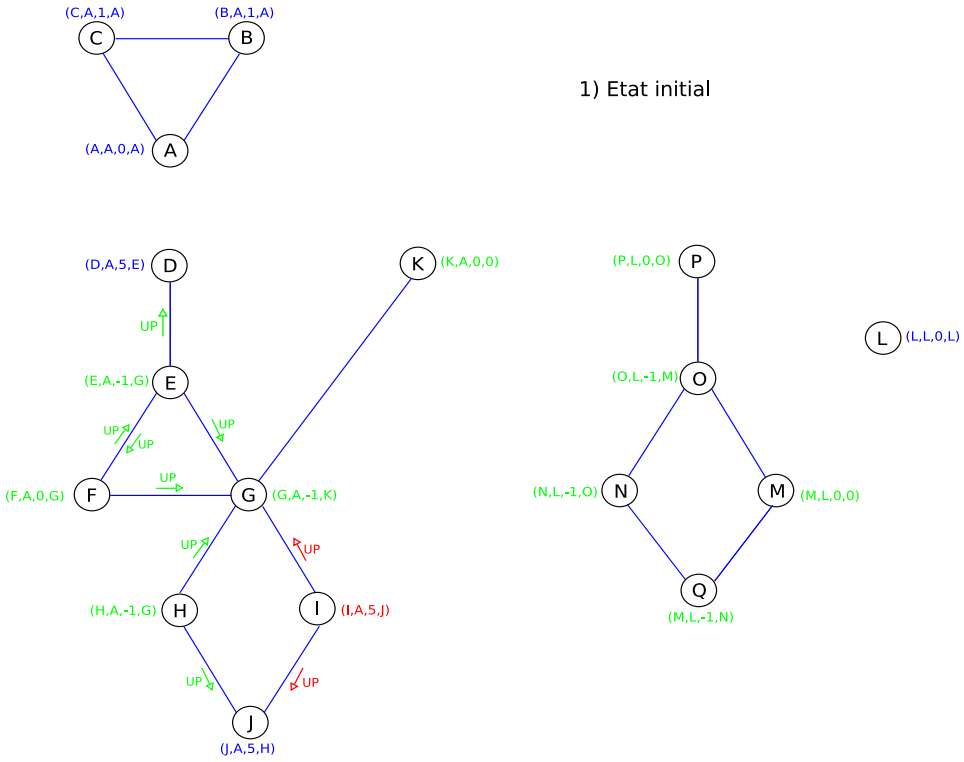


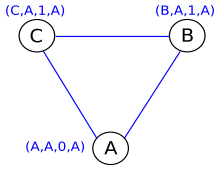
12) En J et D, règle D11 : leur prédécesseur respectif qui était à nul est redevenu défini.



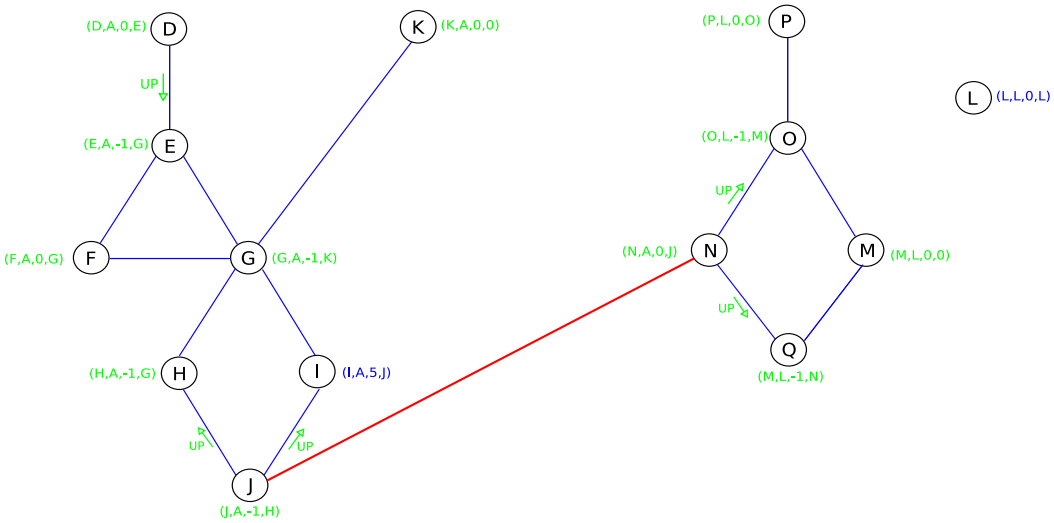
B.4 Connexion - déconnexion

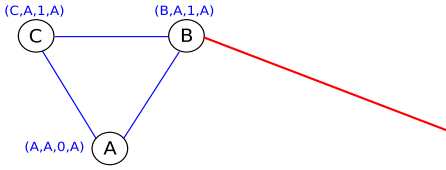
Nous présentons dans cette section l'exemple d'une connexion se faisant entre deux îlots qui sont chacun en recherche d'un chemin vers leur leader. Sur les deux îlots de gauche, les évènements qui se produisent sans lien avec l'îlot de droite sont les mêmes que ceux décrits dans la section précédente (détection d'un îlot). Nous ne les présentons donc pas de nouveau ici.



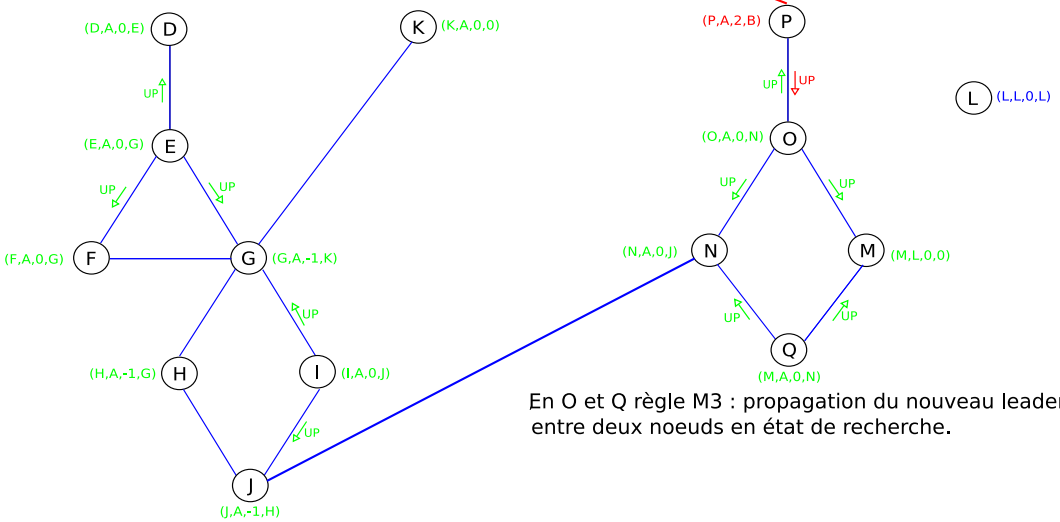


2) En N, règle C3 : le leader de J est gagnant (car A < L) donc N l'adopte et se met à nul car il ne peut offrir aucun chemin vers ce leader.

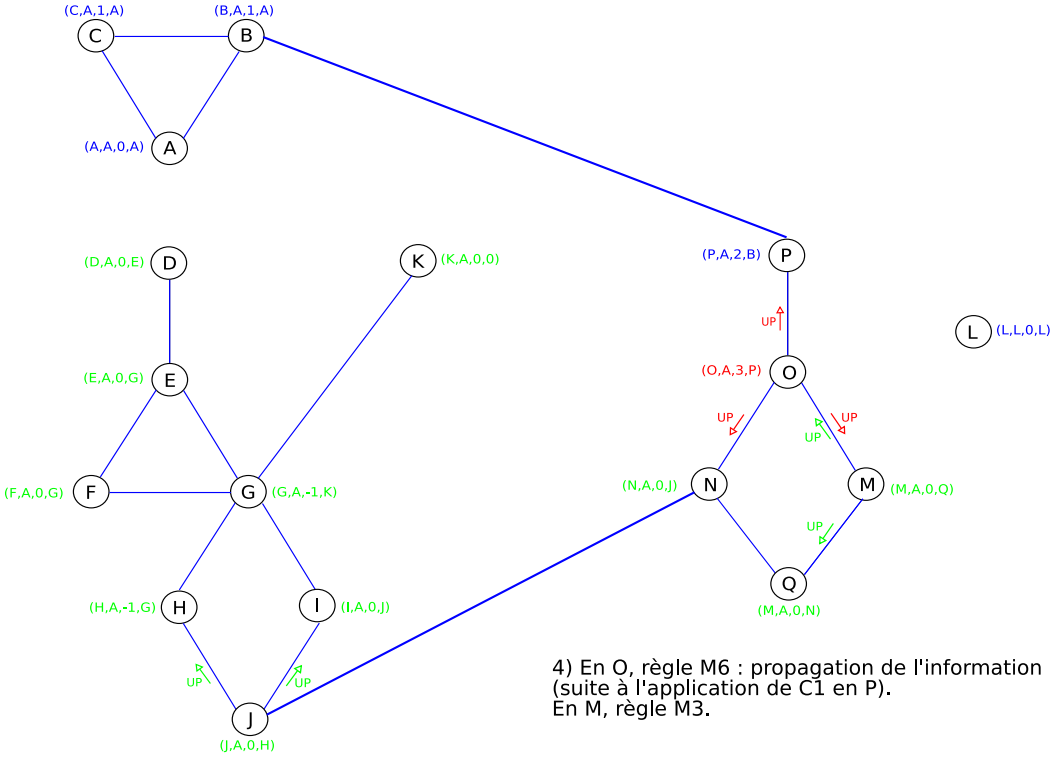




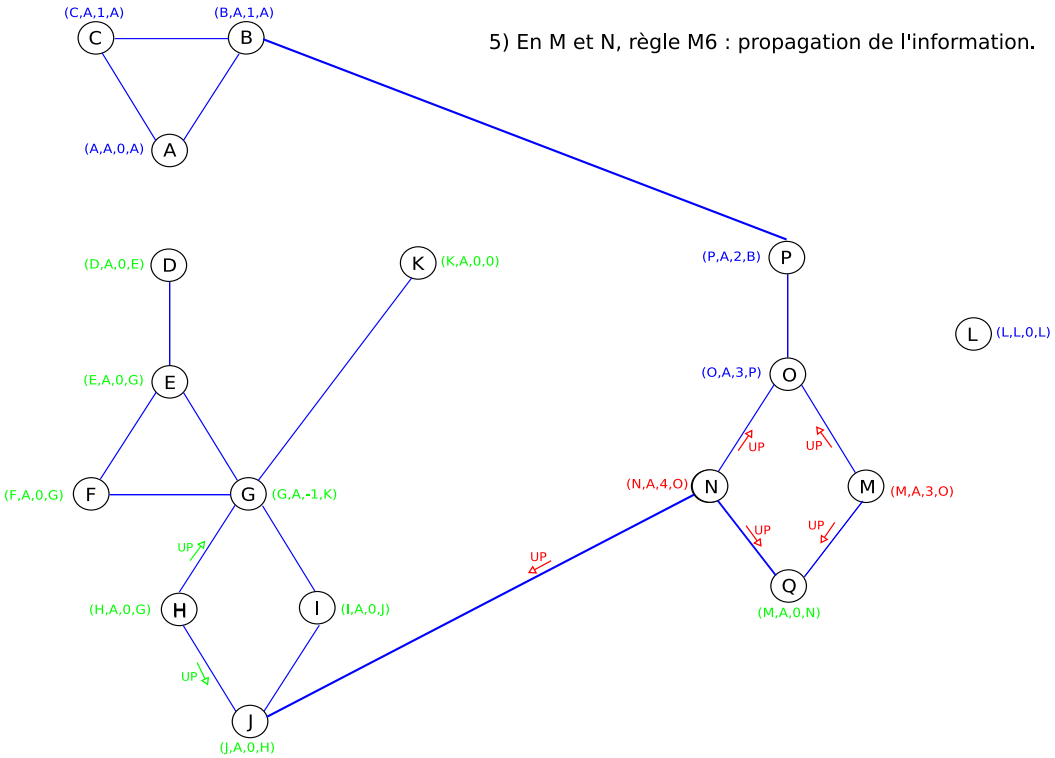
3) En P, règle C1 : P adopte le leader de B car celui-ci est gagnant (A < L) et donc il prend B comme prédécesseur.



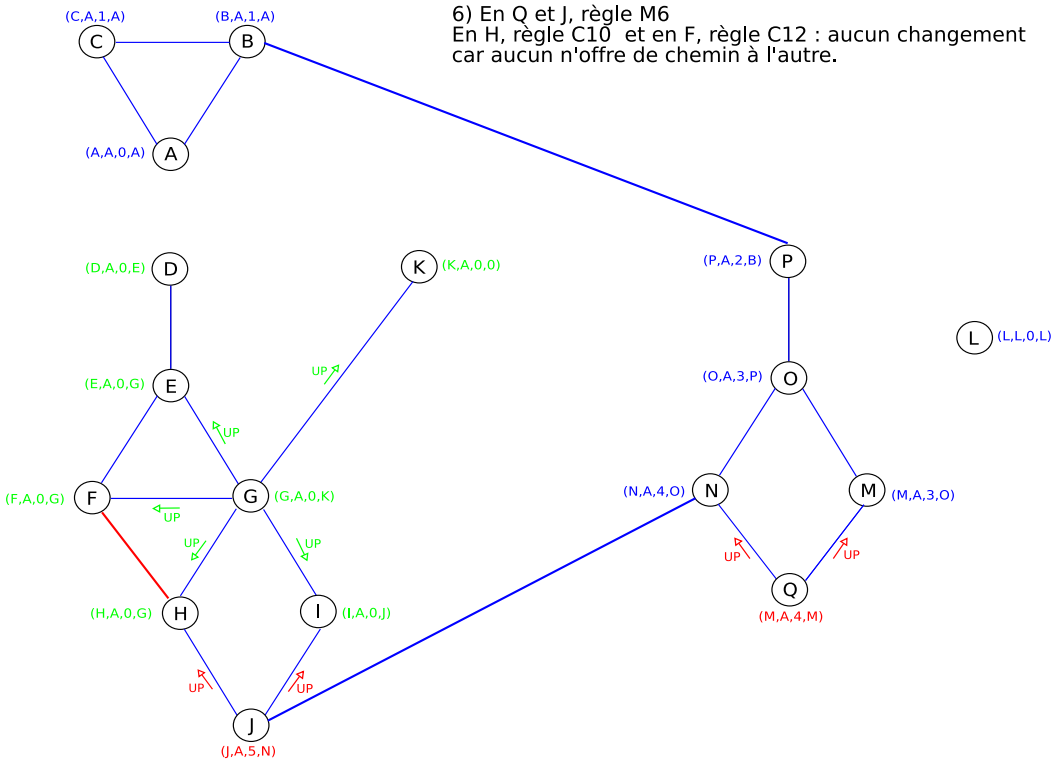
En O et Q règle M3 : propagation du nouveau leader entre deux noeuds en état de recherche.



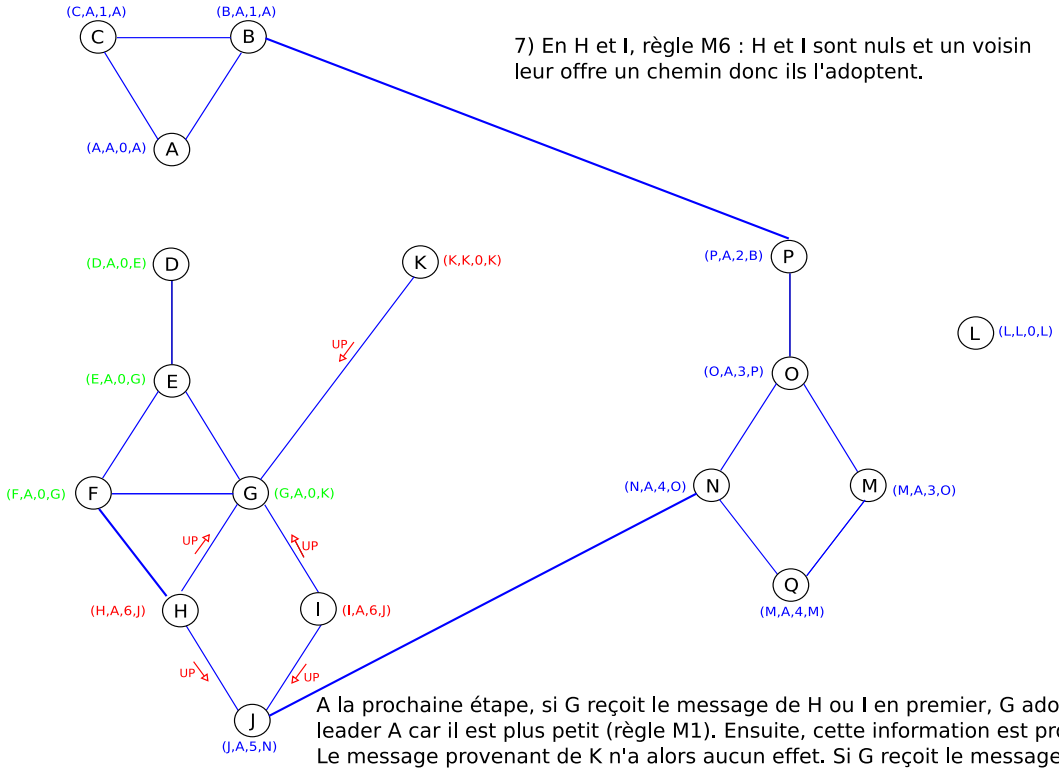
4) En O, règle M6 : propagation de l'information (suite à l'application de C1 en P).
En M, règle M3.



5) En M et N, règle M6 : propagation de l'information.



6) En Q et J, règle M6
En H, règle C10 et en F, règle C12 : aucun changement car aucun n'offre de chemin à l'autre.



7) En H et I, règle M6 : H et I sont nuls et un voisin leur offre un chemin donc ils l'adoptent.

A la prochaine étape, si G reçoit le message de H ou I en premier, G adopte le leader A car il est plus petit (règle M1). Ensuite, cette information est propagée. Le message provenant de K n'a alors aucun effet. Si G reçoit le message de K en premier, G adopte tout d'abord K comme leader selon la règle D11. Ensuite, lorsqu'il reçoit le message de H et I, G adopte A comme leader par la règle M1 (car A est plus petit selon l'odre lexicographique) et l'information se propage.

Annexe C

Validation des protocoles avec AVISPA

Dans cette annexe nous présentons les fichiers AVISPA décrivant nos protocoles.

C.1 Echange de type 1

Un nœud A envoie une requête composée de l'identifiant de requête *Req* et d'un *nonce* *N*. Le récepteur B la traite et renvoie une réponse composée de l'idenfiant de réponse *Resp* correspondant, du *nonce* *N* et d'un message *M* qui doit rester confidentiel. Le premier échange ne contient pas de données critiques donc il est simplement signé. Dans la réponse le message doit rester confidentiel donc il est en plus chiffré.

```
% 1. A -> B : {PKa}SIGNca {Req, N}SIGNa
% 2. B -> A : {PKb}SIGNca {{Resp, N, M}PKa}SIGNb

% les paramètres des rôles sont les éléments visibles par tous,
% c'est à dire :
%    - les identités A (représente Alice) et B (représente Bob) des
%      interlocuteurs
%    - les clés utilisées pour les échanges (PKca et PKa)
%    - les identifiants des canaux de communication

% les variables locales sont le contenu des messages à savoir
% le nonce N, l'OPG, et les identifiants de requête Req et de
% réponse Resp.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
role alice(A, B : agent,
           PKca, PKa, PKb : public_key,
           Hash : hash_func,
           SND,RCV : channel (dy)) played_by A def=

     local
        State   : nat,
        Req, N, Resp, M : text

  init State := 0

  transition
```

```
    step1. State = 0   /\ RCV(start) =|>
           State':= 2  /\ N' := new()
                       /\ Req' := new()

                       /\ SND(PKa.{Hash(PKa)}_inv(PKca).
                              %<------------------------>
                              %signature de la clé publique
                              %propre à la carte

                              Req'.N'.{Hash(Req'.N')}_inv(PKa))
                              %<------------------------------>
                              %       Signature du message

    step2. State = 2   /\  RCV(PKb.{Hash(PKb)}_inv(PKca).{Resp'.N.M'}_PKa.
                                {Hash(Resp'.N.M')}_inv(PKb)) =|>
           State' := 4 /\ request(A,B,alice_bob_n, N)
                          % A doit authentifier le message
                          % grâce à la variable alice_bob_n
end role

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
role bob(A, B : agent,
         PKca, PKa, PKb : public_key,
         Hash : hash_func,
         SND,RCV : channel (dy)) played_by B def=

     local
        State  : nat,
        Req, N, Resp, M : text

  init State := 1

  transition
    step1. State = 1   /\ RCV(PKa.{Hash(PKa)}_inv(PKca).Req'.N'.
                                {Hash(Req'.N')}_inv(PKa)) =|>
           State':= 3  /\ Resp' := new()
                       /\ M' := new()
                       /\ SND(PKb.{Hash(PKb)}_inv(PKca).
                               %<------------------------>
                               %Signature de la clé publique
                               %propre à la carte

                               {Resp'.N.M'}_PKa.{Hash(Resp'.N.M')}_inv(PKb))
                               %<-------------> <--------------------------->
                               %Chiffrement    et     signature du message

                       /\ secret(M, m, {A,B})
                          % l'échange doit assurer que l'élément
                          % opg reste secret
                       /\ witness(B,A,alice_bob_n, N)
                          % le message devra être authentifié
                          % grâce à N
end role

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
role session(A, B: agent, PKca, PKa, PKb: public_key, Hash: hash_func)
```

```
  def=

  local SA, RA, SB, RB: channel (dy)

  composition

        alice(A,B,PKca,PKa,PKb,Hash,SA,RA)
     /\ bob   (A,B,PKca,PKa,PKb,Hash,SB,RB)

end role

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
role environment() def=

    const a, b              : agent,
          pka, pkb, pkca, ki    : public_key,
          h : hash_func,
          alice_bob_n, m: protocol_id

    intruder_knowledge = {a, b, ki, inv(ki),h}

    composition

         session(a,b,pkca,pka,pkb,h)
      /\ session(a,i,pkca,pka,pkb,h)
      /\ session(i,b,pkca,pka,pkb,h)
      /\ session(b,a,pkca,pka,pkb,h)
end role

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
goal

  % Vérification que l'échange concerné conserve la confidentiatité
  % de l'élément opg.
  secrecy_of m
  % Vérification que l'authentification du message de B
  % auprès de A est correcte en considérant l'élément alice_bob_n.
  authentication_on alice_bob_n

end goal

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
environment()

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

C.2 Echange de type 2

Un nœud A envoie à un nœud B un message contenant des données critiques devant rester confidentielles. La requête contient donc ces données ($M1$), l'identifiant de requête *Req* et un *nonce* N. Le message est reçu par B qui le traite et renvoie une réponse contenant elle aussi des données critiques. Cette réponse est donc composée de l'identifiant de réponse *Resp*, du *nonce* N reçu précédemment et des données ($M2$) à envoyer. Les deux échanges

contiennent des données critiques donc ils doivent être signés et chiffrés.

```
% 1. A -> B : {PKa}SIGNca {{Req, N, M1}PKb}SIGNa
% 2. B -> A : {PKa}SIGNca {{Resp, N, M2}PKa}SIGNb

% les paramètres des rôles sont les éléments visibles par tous,
% c'est à dire :
%    - les identités A (représente Alice) et B (représente Bob) des
%      interlocuteurs
%    - les clés utilisées pour les échanges (PKca et PKa)
%    - la clé de groupe dans lequel se passent les communications (PKgr)
%    - les identifiants des canaux de communication

% les variables locales sont le contenu des messages à savoir
% le nonce N, l'OPG, et les identifiants de requête Req et de
% réponse Resp.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
role alice(A, B : agent,
           PKb, PKa,  PKca, PKgr : public_key,
           Hash : hash_func,
           SND,RCV : channel (dy)) played_by A def=

     local
        State  : nat,
        Req, Resp, N, M1, M2: text

  init State := 0

  transition
   step1. State = 0  /\ RCV(start) =|>
          State':= 2 /\ N' := new()
                     /\ Req' := new()
                     /\ M1' := new()
                     /\ SND(PKa.{Hash(PKa)}_inv(PKca).
                            %<--------------------->
                            %Signature de la clé
                            %propre à la carte

                            {Req'.N'.PKgr.PKa.M1'}_PKb.
                            %<--------------------->
                            %Chiffrement du message

                            {Hash(Req'.N'.PKgr.PKa.M1')}_inv(PKa))
                            %<------------------------------------>
                            %          Signature du message

                     /\ secret(M1, m1, {A,B})
                        % l'échange doit assurer que l'élément
                        % m1 reste secret

    step2. State = 2  /\ RCV(PKb.{Hash(PKb)}_inv(PKca).
                             {Resp'.N.M2'.PKgr}_PKa.
                             {Hash(Resp'.N.M2'.PKgr)}_inv(PKb)) =|>
           State':= 4 /\ request(A,B,alice_bob_n,N)
                         % A doit authentifier le message
                         % grâce à N
end role
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
role bob(A, B : agent,
          PKb, PKa,  PKca, PKgr : public_key,
          Hash : hash_func,
          SND,RCV : channel (dy)) played_by A def=

     local
        State  : nat,
        Req, Resp, N, M1, M2: text

  init State := 1

  transition
    step1. State = 1  /\ RCV(PKa.{Hash(PKa)}_inv(PKca).
                            {Req'.N.PKgr.PKa.M1'}_PKb.
                            {Hash(Req'.N.PKgr.PKa.M1')}_inv(PKa)) =|>

          State':= 3 /\ Resp' := new()
                     /\ M2' := new()
                     /\ SND(PKb.{Hash(PKb)}_inv(PKca).
                            %<------------------------->
                            %signature de la clé
                            %propre à la carte

                            {Resp'.N.M2'.PKgr}_PKa.
                            %<---------------------->
                            %Chiffrement du message

                            {Hash(Resp'.N.M2'.PKgr)}_inv(PKb))
                            %<--------------------------------------->
                            %         Signature du message

                     /\ witness(B,A,alice_bob_n,N)
                         % A doit authentifier le message
                         % grâce à N
end role

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
role session(A, B: agent, PKb, PKa,  PKca, PKgr: public_key,
  Hash: hash_func) def=

  local SA, RA, SB, RB: channel (dy)

  composition

        alice(A,B,PKb,PKa,PKca,PKgr,Hash,SA,RA)
     /\ bob  (A,B,PKb,PKa,PKca,PKgr,Hash,SB,RB)

end role

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
role environment() def=

    const a, b          : agent,
    pkb, pka, pkca, pkgr   : public_key,
```

```
    h : hash_func,
    alice_bob_n, m1: protocol_id

    intruder_knowledge = {a, b, pkgr, h}

    composition

        session(a,b,pkb,pka,pkca,pkgr,h)
     /\ session(a,i,pkb,pka,pkca,pkgr,h)
     /\ session(i,b,pkb,pka,pkca,pkgr,h)

end role

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
goal

  % Vérification que l'échange concerné conserve la confidentiatité
  % de l'élément opg.
  secrecy_of m1
  % Vérification que  l'authentification du message  de B
  % auprès  de A est correcte en considérant l'élément alice_bob_n.
  authentication_on alice_bob_n

end goal

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
environment()

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

C.3 Echange de type 3

Dans ce type d'échange il s'agit simplement pour un nœud A d'envoyer dans une requête d'identifiant *Req* un message *M* devant rester confidentiel. Pour garder *M* secret, l'échange doit être chiffré en plus d'être signé. Le message étant adressé à tous les nœuds du groupe *gr*, il est chiffré avec la clé *PKgr* de ce groupe.

```
% 1. A -> B : {PKa}SIGNca {{Req, PKgr, M}PKgr}SIGNa

% les paramètres des rôles sont les éléments visibles par tous,
% c'est à dire :
%    - les identités A (représente Alice) et B (représente Bob) des
%      interlocuteurs
%    - les clés utilisées pour les échanges (PKca et PKa)
%    - la clé de groupe dans lequel se passent les communications (PKgr)
%    - les identifiants des canaux de communication

% les variables locales sont le contenu des messages à savoir
% le nonce N, l'OPG, et les identifiants de requête Req et de
% réponse Resp.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
role alice(A, B : agent,
           PKb, PKa, PKca, PKgr : public_key,
           Hash : hash_func,
```

```
            SND,RCV : channel (dy)) played_by A def=

      local
         State  : nat,
         Req, Resp, N, M: text

   init State := 0

   transition
     step1. State = 0   /\ RCV(start) =|>
            State':= 2  /\ N' := new()
                        /\ Req' := new()
                        /\ M' := new()
                        /\ SND(PKa.{Hash(PKa)}_inv(PKca).
                               {Req'.PKgr.M'}_PKgr.
                               %<------------------>
                               %  Chiffrement et...

                               {Hash(Req'.PKgr.M')}_inv(PKa))
                               %<------------------------------>
                               %    ...signature du message

                        /\  secret(M, m, {A,B})
                           % l'échange doit assurer que l'élément
                           % M reste secret

end role

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
role bob(A, B : agent,
            PKb, PKa, PKca, PKgr : public_key,
            Hash : hash_func,
            SND,RCV : channel (dy)) played_by A def=

      local
         State  : nat,
         Req, Resp, N, M : text

   init State := 1

   transition
     step1. State = 1   /\ RCV(PKa.{Hash(PKa)}_inv(PKca).{Req'.PKgr.M'}_PKgr.
                               {Hash(Req'.PKgr.M')}_inv(PKa)) =|>

            State':= 3 % On passe à l'état suivant pour la réception
                       % mais bob ne renvoie rien.
end role

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
role session(A, B: agent, PKb, PKa, PKca, PKgr: public_key,
  Hash: hash_func) def=

   local SA, RA, SB, RB: channel (dy)

   composition
```

```
        alice(A,B,PKb,PKa,PKca,PKgr,Hash,SA,RA)
     /\ bob   (A,B,PKb,PKa,PKca,PKgr,Hash,SB,RB)

end role

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
role environment() def=

    const a, b           : agent,
    pkb, pka, pkca, pkgr    : public_key,
    h : hash_func,
    alice_bob_n, m: protocol_id

    intruder_knowledge = {a, b, pkgr, h}

    composition

        session(a,b,pkb,pka,pkca,pkgr,h)
     /\ session(a,i,pkb,pka,pkca,pkgr,h)
     /\ session(i,b,pkb,pka,pkca,pkgr,h)

end role

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
goal

  % Vérification que l'échange concerné conserve la confidentiatité
  % de l'élément opg.
  secrecy_of m

end goal

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
environment()

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```