Université Abou Bekr Belkaid
Tlemcen Algérie
جامعة أبي بكر بلقايد
تلمسان الجزائر

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Laboratoire STIC

THESE

Présentée

A L'UNIVERSITE DE TLEMCEN
FACULTE DE TECHNOLOGIE

Pour l'obtention du diplôme de

DOCTORAT

Spécialité : " Télécommunications"

Par

ABDELMALEK Abdelhafid

SECURITE DES RESEAUX SANS FIL HAUTS DEBITS: APPLICATION A L'INITIALISATION ET L'AUTO-CONFIGURATION DES RÈSEAUX MOBILES ADHOC

Soutenue en 2013 devant le Jury:

CHIKH Mohamed Amine	Pr, Université de Tlemcen	Président
HAFFAF Hafid	Pr, Université d'Oran	Examinateur
SEKHRI Larbi	MCA, Université d'Oran	Examinateur
DIDI Fedoua	MCA, Université de Tlemcen	Examinateur
FEHAM Mohammed	Pr, Université de Tlemcen	Directeur de Thèse
TALEB-AHMED Abdelmalik	Pr, Université de Valenciennes (France)	Co-directeur de Thèse

Remerciements

Ce mémoire de thèse est la concrétisation d'un travail de recherche de plusieurs années où certainement plusieurs acteurs en y participer. Je voudrais remercier ici toutes les personnes qui ont contribué directement ou indirectement à l'aboutissement de cette thèse.

Mes plus grands remerciements sont naturellement pour le professeur Mohammed FEHAM directeur du laboratoire STIC, qui m'a encadré tout au long de ma thèse. Ses remarques et critiques pertinentes m'ont conduit vers la bonne voie. Sa rigueur, sa passion pour la recherche et son soutien m'ont permis de ne jamais faiblir et de poursuivre toujours plus loin mes travaux. Ma considération est inestimable pour ses recommandations et pour la confiance qu'il m'a témoignée dans de nombreuses circonstances.

Ma reconnaissance se tourne particulièrement vers le professeur Taleb-Ahmed Abdelmalik de l'université de Valenciennes et du Hainaut-Cambrésis, pour m'avoir accueilli, dans le cadre de ma formation alternée, au sein du laboratoire LAMIH et mis à ma disposition tous les moyens scientifiques nécessaires pour finaliser cette thèse.

J'exprime ma gratitude au professeur CHIKH Mohamed Amine de l'université de Tlemcen pour l'honneur qu'il me fait en présidant mon Jury de thèse, ainsi qu'au professeur HAFFAF Hafid et SEKHRI Larbi Maitre de conférence de l'université d'Oran, et Madame DIDI Fedoua Maitre de conférence de l'université de Tlemcen, pour l'honneur qu'ils me font en participant à mon jury. Je les remercie sincèrement pour le temps qu'ils ont consacré à la lecture, à l'évaluation de mon travail et à la rédaction des rapports d'expertise.

Enfin, j'ai une chaleureuse pensée pour tous mes collègues de travail pour leur aide, la bonne ambiance qu'ils ont pu apporter et la qualité des relations professionnelles et amicales que nous avons tissées ces dernières années.

Résumé

Dans le cadre de la sécurité des réseaux mobiles ad hoc (MANETs), la plupart des recherches jusqu'à présent ont porté principalement sur les modèles de confiance, les problèmes de sécurité de routage et beaucoup moins d'attention a été accordée à la question de la sécurité d'auto-configuration. Cependant, le manque de sécurité dans la conception de tous les protocoles d'auto-configuration précédemment proposés ouvre la possibilité de nombreuses menaces réelles, en raison des vulnérabilités des réseaux MANETs bien connues et qui sont spécifiques au paradigme ad hoc. En conséquence, cela peut conduire dans des environnements potentiellement hostiles à des attaques sérieuses. Quelques mécanismes ont été proposés pour résoudre ce problème, mais aucun d'eux n'a apporté de solutions satisfaisantes. La sécurité de l'auto-configuration est toujours un problème ouvert. Dans cette thèse, un protocole d'auto-configuration d'adresses IP sans état, robuste et sécurisé pour les réseaux MANETs autonomes, est spécifié et évalué sous NS2. Notre solution est basée sur une authentification mutuelle et un modèle de confiance d'Autorité de Certification et d'Auto-configuration complètement distribué, utilisant le support de la cryptographie à seuil basée sur le problème du logarithme discret (DLP). En déployant un nouveau concept de certificat conjointe liant l'adresse IP à la clé publique, nous résolvons irrévocablement le problème de quelques attaques comme IP spoofing et l'attaque Sybile, sans solution à ce jour par les mécanismes conventionnels. Notre protocole nommé TCSAP (Threshold Cryptography based Secure Auto-configuration Protocol) fournit à la fois la sécurité et la robustesse et surmonte toutes les limites des approches proposées précédemment, tout en assurant l'allocation dynamique des adresses IP en temps opportun.

Mots clés : *TCSAP, Cryptographie à seuil, Logarithme discret , partage de secret, réseaux mobiles adhoc, Auto-configuration, Joint IP address and Public key Certificate, signature partielle ElGamal, NS2.*

Table des matières

Titre…...……..…i

Résumé……..…..……...……...…iv
Table des matières..……...…v
Liste des figues……..…..……..……... ix
Liste des tableaux……..……..……..……..xi
Acronymes…..……..……..xii

Introduction générale……...……..1

CHAPITRE I

L'auto-configuration dans les réseaux MANETS

Etat de l'art

I.1 Introduction.....6
I.2 Contraintes des réseaux MANETs 7
I.3 Exigences des schémas d'auto-configuration 7
I.4 Classification des approches d'auto-configuration MANETs proposées.....8
I.4.1 Approches Stateful.....8
I.4.1.1 Approches stateful avec reproduction totale de l'information d'état.....8
a- MANETconf.....8
I.4.1.2 Approches stateful sans reproduction de l'information d'état.....10
a- Binary Split.....10
b- Prophet Allocation.....11
I.4.1.3 Approches stateful avec reproduction partielle de l'information d'état.....13
a- Coor-Root 13
b- Approche quorum based 14
I.4.2 Approches Stateless 16
a- Approche Strong DAD.....16
b- Approche Strong-Weak DAD 17
I.4.3 Solutions Hybrides 18
a- Approche PACMAN (Passive DAD).....18
b- Approche Leader based.....19
I.5 Comparaison qualitative des approches d'auto-configuration 21
I.6 Conclusion.....23

CHAPITRE II

Sécurité de l'auto-configuration des réseaux MANETs Menaces réelles et exigences

II.1 Introduction 24
II.2 Menaces de sécurité 25
II.2.1 Attaque d'usurpation d'adresse IP (IP Spoofing) 25
II.2.2 Attaque d'épuisement de l'espace d'adressage (Exhaustion Address Space) 25
II.2.3 Attaque de conflit d'adresses (Conflict Address) 25
II.2.4 Attaque de faux conflit d'adresses (False Conflict Address) 25
II.2.5 Attaque sybile 26
II.2.6 Attaque de DoS (surcharge du réseau) 26
II.3 Aperçu et analyse des schémas de sécurité précédents 26
II.3.1 Autorité de certification on-line à un saut 27
II.3.2 Pré-authentification et autorisations à seuil 27
II.3.3 Adresse IP basée sur le haché de la clé publique 28
II.3.4 Modèle de sécurité basé sur une valeur de confiance 29
II.3.5 Signature des messages de l'autorité d'adresse 30
II.3.6 Secure Prophet 31
II.4 Conclusion sur l'analyse des schémas de sécurité précédents 32
II.5 Exigences de sécurité pour le service d'auto-configuration 33
II.6 Conclusion 35

CHAPITRE III

Modèles de confiance de PKI distribuée et Cryptographie à seuil

III.1 Introduction 36
III.2 Modèles de Confiance basés sur la cryptographie asymétriques 37
III.2.1 Modèle de PKI basée sur CA distribuée 38
a- Modèle de CA partiellement distribuée 38
b- Modèle de CA totalement distribuée 40
III.2.2 Modèle de certificat basé sur l'identité 41
III.2.3 Modèle de clé publique auto-certifiée 42
III.2.4 Modèle de PKI auto-organisée 42
III.2.5 Discussion 43
III.3 Généralités sur la Cryptographie à seuil 44
III.3.1 Bases de la théorie des nombres appliquée à la cryptographie asymétrique 44
III.3.2 Théorie de la complexité 46
III.3.3 Modèle de sécurité des schémas cryptographiques à seuil 47
III.4 Problèmes à résoudre pour les schémas cryptographiques à seuil 50

III.4.1 Fraude du distributeur : protocoles VSS et PVSS 50
III.4.2 Nécessité de protocoles distribués 51
III.4.3 Attaques répétées dans le cas d'un MANET à vie longue : Partage de secret proactif 51
III.5 Partage de secret à seuil 51
III.5.1 Partage de secret à seuil (k,n) de Shamir 52
III.6 Partage de fonction à seuil 53
III.6.1 Schémas de signature à seuil 53
III.7 Conclusion 54

CHAPITRE IV

Protocole TCSAP
Description et Spécification

IV.1 Introduction 55
IV.2 Modèle de confiance à seuil: Module DPKI 56
IV.2.1 Environnement cryptographique 57
IV.2.2 Partage de secret de Shamir 57
IV.2.3 Protocole distribué et vérifiable de partage de secret aléatoire 58
IV.2.4 Attribution d'une part du secret à un nouveau nœud 60
IV.2.5 Signature à seuil de type ElGamal 61
IV.2.5.1 Schéma de signature ElGamal 61
IV.2.5.2 Schéma de signature DSA 61
IV.2.5.3 Schéma de signature variante de DSA 62
IV.2.5.4 Schéma de signature à seuil basé sur la variante de DSA 62
IV.3 Modèle d'auto-configuration à seuil : Module TCSAP 63
IV.3.1 Information d'état maintenue 63
IV.3.2 Attribution d'une adresse IP 65
IV.4 Certificat commun de clé publique et d'adresse IP 66
IV.4.1 Modification du Certificat X.509 66
IV.5 Hypothèses 67
IV.6 Spécification du protocole TCSAP 67
IV.6.1 Etats d'un nœud 69
IV.6.2 Format générique des paquets TCSAP 69
IV.6.3 Protocole de Découverte des voisins 70
IV.6.4 Protocole d'initialisation du réseau MANET 73
IV.6.5 Protocole d'auto-configuration d'un nouveau nœud 78
IV.6.6 Scénarios de mobilité et changement de topologie 89
IV.7 Analyse de sécurité 91
IV.8 Conclusion 93

CHAPITRE V

Evaluation des Performances du Protocole TCSAP

V.1 Introduction 95
V.2 Méthodologie d'évaluation 95
V.2.1 Outil de simulation 96
V.2.2 Une évaluation de deux composantes de TCSAP 97
V.2.3 Critères d'évaluation 97
V.2.4 Paramètres du protocole 98
V.3 Implémentation du protocole TCSAP 98
V.4 Simulations 102
V.4.1 Scénarios de simulations 103
IV.4.1.1 Etude 1 : Auto-initialisation d'un réseau MANET 103
IV.4.1.2 Etude 2 : Auto-configuration d'un nouveau nœud joignant le réseau MANET 103
V.5 Résultats et Observations 106
V.5.1 Résultats de l'étude 1 : Auto-initialisation 106
V.5.2 Résultats de l'étude 2 : Auto-configuration 107
IV.5.2.1 Impact de la densité du réseau 107
IV.5.2.2 Impact du taux d'arrivée des nœuds 111
IV.5.2.3 Impact de la taille du réseau 111
IV.5.2.4 Impact de la mobilité 113
IV.5.2.5 Impact de la charge du réseau 114
V.6 Conclusion 115

Conclusion générale 117
Bibliographie 122
Annexe A : IMPLÉMENTATION DU PROTOCOLE TCSAP (CODE C++) 128
Annexe B : SCRIPTS DES SIMULATIONS SOUS NS2 (CODE TCL) 168

Liste des figures

Figure **page**

Figure 1.1 - Exemple de topologie à vote quorum ……..…….....….. 15

Figure 4.1 – Division de l'espace d'adressage et 'Free IP Address Table'….……..... 64

Figure 4.2 - Format du certificat X.509.....…… 67

Figure 4.3 - Blocs fonctionnels des modules DPKI et TCSAP …...……. 69

Figure 4.4 - Format générique des paquets TCSAP.....…… 70

Figure 4.5 - Format du paquet Discovery_Request ……...…….....… 70

Figure 4.6 - Format du paquet Discovery_Reply…….……..… 71

Figure 4.7 - Format du paquet Discovery_Welcome ……...……..... 71

Figure 4.8 - Algorithme pour le protocole de découverte de voisins MANET_NDP……... 74

Figure 4.9 - Initialisation d'un réseau MANET avec k = 4…….…...……... 74

Figure 4.10 - Format du paquet Init_Start …….....……..….. 75

Figure 4.11 - Format du paquet Init_Advert .....…….....…… 75

Figure 4.12 - Format d'adresse IPv6 de type « site-local »...……...…… 76

Figure 4.13 - Algorithme pour le protocole d'initialisation MANET_BSP ...…….…. 79

Figure 4.14 - Allocation d'adresse IP (Expanding ring search)……..……. 80

Figure 4.15 - Format du paquet Config_Request ……..…….....…. 80

Figure 4.16 - Format du paquet Config_Replay…….....……. 81

Figure 4.17 - Format du paquet Config_Cert_Request .....…….. 81

Figure 4.18 - Format du paquet Config_Cert_Replay …….....…….. 82

Figure 4.19 - Format du paquet Config_Info .....…… 82

Figure 4.20 - Algorithme de recherche des serveurs.....…… 84

Figures **page**

Figure 4.21 - Algorithme de traitement d'un message Config_Request par un serveur ……..…... 85

Figure 4.22 - Algorithme de choix des co-signataires …… 86

Figure 4.23 - Traitement d'un message Config_Cert_Request par un cosignataire 87

Figure 4.24 - Algorithme d'enregistrement d'un nouveau nœud et résolution de conflit d'adresses..... 88

Figure 4.25 - Algorithme de traitement d'une demande d'enregistrement.....……..... 88

Figure 5.1 - Intégration de l'agent TCSAP dans NS2.....…….... 99

Figure 5.2 - Evolution du délai d'auto-configuration en fonction du seuil k……..... 106

Figure 5.3 - Mesure de la valeur optimale du paramètre ISTR_TIMEOUT en fonction du seuil 107

Figure 5.4 - Evolution du délai d'auto-configuration en fonction du seuil k .....…….... 108

Figure 5.5 - Degré d'un nœud en fonction de la densité .....…….... 108

Figure 5.6 - Mesure de la valeur optimale du paramètre CREQ_TIMEOUT en fonction du seuil k 109

Figure 5.7 - Mesure de la valeur optimale du paramètre CCREQ_TIMEOUT en fonction du seuil k 109

Figure 5.8 - Mesure de la valeur optimale du paramètre rk en fonction de la densité…….. 110

Figure 5.9 - Overhead de communication en fonction de la densité et du seuil k……. 110

Figure 5.10 - Impact du taux d'arrivée des nœuds sur la probabilité de collision d'adresses 111

Figure 5.11 - Evolution du délai d'enregistrement en fonction de la taille du réseau....……. 112

Figure 5.12 - Exemple de format de trame Evolution de l'overhead de communication pendant la phase d'enregistrement en fonction de la taille du réseau 112

Figure 5.13 - Evolution du délai d'auto-configuration en fonction de la mobilité..... 113

Figure 5.14 - Evolution de l'overhead de communication en fonction de la mobilité …….. 114

Figure 5.15 - Evolution du délai d'auto-configuration en fonction de la charge du réseau 114

Figure 5.16 - Impact de la taille du réseau (référence [51])…….....…… 115

Figure 5.17 - Impact de mobilité (référence [51]).....……. 115

Liste des tableaux

Tableau	Page
Tableau 1.1 - Comparaison qualitative des performances des approches d'autoconfiguration pour les réseaux MANETs	22
Tableau 2.1 - Attaques possibles sur quelques schémas d'auto-configuration pour les réseaux MANETs	26
Tableau 2.2 - Récapitulatif des faiblesses et limitations des propositions existantes	33
Tableau 4.1 - Table RAT (Registered IP Address Table)	65
Tableau 5.1 - Paramètres de simulation	102
Tableau 5.2- Paramètres de variation de la densité du réseau	104
Tableau 5.3- Paramètres du protocole TCSAP (variation de la densité)	104
Tableau 5.4- Paramètres de variation de la taille du réseau	104
Tableau 5.6- Paramètres du protocole TCSAP (variation de la taille)	104
Tableau 5.7- Paramètres de simulation (variation de la charge du réseau)	105

Liste des Acronymes

AA	Address Authority
ANL	Accused Nodes List
AODV	Ad hoc On Demand Distance Vector routing protocol
ARAN	Authenticated Routing protocol for Ad hoc Networks
BL	Black List
CA	Certificate Authority
COCA	Cornell Online Certification Authority
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
DACP	Dynamic Address Configuration Protocol
DAD	Duplicate Address Detection
DCDP	Dynamic Configuration and Distribution Protocol
DH	Diffie-Hellman
DHCP	Dynamic Host Configuration Protocol
DHP	Diffie-Hellman Problem
DLP	Discrete Logarithm Problem
DoS	Denial of Service
DPKI	Distributed Public Key Infrastructure
DSA	Digital Signature Algorithm
DSDV	Distance Source Distance Vector routing protocol
DSR	Distance Source Routing
ECDLP	Elliptic Curve Discrete Logarithm Problem
FAT	Free IP Address Table
FP	Factorization Problem
IBC	Identity Based Cryptography
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPv6	Internet Protocol version 6
L3	Layer 3
MAC	Media Access Control
MAE	MANET Authentication Extension
MANET	Mobile Ad hoc Networks
MANET_BSP	MANET BootStrapping Protocol
MANET_NDP	MANET Neighbors Discovery protocol
MD5	Message-Digest Algorithm
MOCA	Mobile Certificate Authority
MP	MOCA Certification Protocol

MPC	MultiParty Computation
NODE_ACP	NODE Auto-Configuration Protocol
NP	Nondeterministic Polynomial-time
NS2	Network Simulator 2
ODACP	Optimized Dynamic Address Configuration Protocol
OLSR	Optimized Link State Routing protocol
P	Polynomial-time
PACMAN	Passive Auto-configuration for Mobile Ad hoc Networks
PAT	Pending IP Address Table
PGP	Pretty Good Privacy
PKI	Public Key Infrastructure
PVSS	Publicly Verifiable secret Sharing
QoS	Quality of Service
RAT	Registered IP Address Table
RC	Requester Counter
RCL	Revoked Certificates List
RFC	Request for Comment
RSA	Rivest, Shamir and Adleman
SAODV	Secure Ad hoc On-demand Distance Vector
SEAD	Secure Efficient distance vector routing for mobile wireless AD hoc networks
SHA-1	Secure Hash Algorithm 1
SOLSR	Secure Optimized Link State Routing protocol
SRP	Secure Routing Protocol
SSS	Secret Sharing Scheme
TBRTF	Topology Dissemination Based one acts Reverse-Path Forwarding
TCP	Transport Control Protocol
TCSAP	Threshold Cryptography based Secure Autoconfiguration Protocol
UUID	Universal Unique ID
UWB	Ultra Wide Band
VANET	Vehicular Area Network
VSS	Verifiable secret Sharing

Introduction générale

Contexte

Dans le cadre général de la croissance qu'ils ont connue au cours des dernières années, les réseaux sans fil ont suscité un intérêt particulier qui a favorisé un important développement scientifique et technique. Une réelle volonté de développement, d'intégration et de standardisation des réseaux mobiles ad hoc en particulier, a entraîné en 1995 la création au sein de l'IETF du groupe de travail MANET (Mobile Ad hoc Networks).

Les réseaux MANETs sont par nature des réseaux sans fil autonomes c'est-à-dire sans aucune infrastructure. La topologie est dynamique du fait que les nœuds formant le réseau sont mobiles et peuvent donc joindre ou quitter le réseau spontanément. Ces nœuds peuvent, en outre, être uni/multi-interfaces radio de type 802.11b/g/a/n, hyperLan, UWB, ou autres.
Les réseaux MANETs sont très attractifs pour les applications civiles et militaires dans les environnements où les infrastructures sans fil sont difficilement envisageables voire impossibles (champ de bataille, opération logistique, catastrophe naturelle : coordination des secours, réseaux de véhicules (VANET, conférence, réunion, …).

Beaucoup de travaux de recherche ont été entrepris pour résoudre les problèmes inhérents aux réseaux MANETs en l'occurrence le routage, l'auto-configuration, la sécurité, etc.

Concernant le routage, pour assurer cette fonctionnalité en absence de toute infrastructure, les nœuds dans le réseau MANET doivent faire office de routeurs. Il y a donc nécessité de développer de nouveaux protocoles de routage multi-sauts (multihops). Les travaux du groupe de travail WG-MANET chargé de définir de nouveaux standards pour les protocoles de routage spécifiques aux réseaux mobiles ad hoc ont conduit à la publication de deux familles de protocoles : les protocoles réactifs (recherche de route à la demande, eg. AODV [1], DSR [2]) et les protocoles proactifs

(maintien continu de tables de routage, eg. OLSR [3], TBRPF [4]). D'autres publications ayant trait à l'étude de nouvelles approches pour le routage ad hoc existent, concernant notamment les protocoles hybrides [5] et les protocoles géographiques [6].

D'autre part, l'absence dans les réseaux MANETs d'une infrastructure dotée d'un serveur DHCP permettant l'assignation automatique d'adresses IP implique la nécessité de développement de protocoles d'auto-configuration distribués. A ce sujet, plusieurs solutions ont été également proposées, mais aucune d'elles n'a fait l'objet d'un standard. Ces solutions peuvent être classées selon les trois approches suivantes :

- Les approches stateful (à information d'état) nécessitant le maintien de tables d'allocation d'adresses IP. Il existe trois variantes selon que la reproduction de l'information d'état est totale (MANETConf [7]), partielle (Quorum based [8]) ou nulle (Binary split [9], Prophet [10]).

- Les approches stateless (sans information d'état) qui reposent sur une allocation aléatoire et des mécanismes de vérification de l'unicité d'adresses (DAD : Duplicate Address Detection) [11].

- Les approches hybrides combinant les mécanismes stateful et stateless [12,13].

D'un point de vue sécurité, les réseaux MANETs sont très vulnérables à cause des liens radio ouverts. Le contexte donc de la sécurité demeure un point crucial. En raison des contraintes liées à la mobilité, aux liens radio (débit, qualité, ..) et aux ressources physiques limitées (capacité de calcul et de stokage, et réserve d'énergie), les réseaux MANETs possèdent des exigences spécifiques en termes de sécurité.

Les services de sécurité exigés se résument dans les points suivants :

- La confidentialité des données circulant sur le réseau.
- L'intégrité qui assure que les données n'ont pas été modifiées.
- L'authentification des nœuds participants au réseau.
- Non répudiation pour isoler les nœuds malicieux ou compromis.
- La disponibilité qui peut être vue comme l'intégrité du réseau contre toute attaque de déni de service.

Les différents aspects de sécurité traités par la littérature se résument dans les points suivants :

- Le premier point concerne la définition de modèles de confiance et la mise au point de mécanismes d'authentification et de gestion des clés (Key managment). Diverses approches symétriques (Resurrecting Duckling Model [14], Key Agreement Model [15], Pairewise Key Predistribution Model [16]) et asymétriques (Distributed CA Model [17], Identity Based

Model [18], Self Certified Public Key Model [19], Self Organization Model [20]) ont été proposées pour la base de confiance, avec des avantages et des inconvénients selon le domaine d'applications.

- Le second point concerne la sécurité des protocoles de routage. Les solutions proposées pour les protocoles de routage supposent que tous les nœuds sont dignes de confiance et collaborent automatiquement. Or cela est loin d'être le cas, les nœuds peuvent être malicieux ou égoïstes. A ce niveau, plusieurs attaques ont été mises en évidence telles que : trou noir, trou gris, trou de ver, attaque sybile, attaque de rejeu, empoisonnement de table de routage, déni de service (torture, débordement de table,...). Pour contrer ces attaques, plusieurs solutions ont été proposées aussi génériques telles que ARAN [21] et SRP [22], ou spécifiques à chaque protocole : SEAD [23] pour DSDV, Ariadne [24] pour DSR, SAODV [25] et SOLSR [26]. Ces solutions se basent sur l'utilisation des signatures, de l'horodatage et les numéros de séquence.

- Le troisième point traité concerne la sécurité des données utiles (chiffrement, intégrité et non répudiation).

Il est communément admis par les spécialistes du domaine que si le premier point est garanti alors les deux autres sont facilement solubles en utilisant des primitives comme la signature et le chiffrement.

Cependant, malheureusement dans le cas de la sécurisation des protocoles de routage, tous les travaux proposés font l'hypothèse d'un modèle de confiance préalablement établi. Or ceci pose un problème récursif dans le sens où l'authentification et la gestion des clés repose elle-même sur des protocoles de routage sécurisés.

Ceci nous amène à la première issue de sécurité traitée dans cette thèse : l'initialisation de la sécurité dans un réseau MANET (security bootstrapping) sans partie tierce. En effet, étant donné un réseau MANET isolé (standalone) et un modèle de confiance, quelle est la solution permettant, sans l'appui d'un protocole de routage, d'initialiser tous les nœuds avec les paramètres de sécurité nécessaires permettant la mise en place de la base de confiance et le maintien des services de sécurité associés ?

Une seconde issue encore ouverte est la sécurité de l'auto-configuration. Plusieurs failles des protocoles d'auto-configuration proposés ont été découvertes et des attaques possibles ont été mises en évidence [27-33] ; Les propositions publiées pour contrecarrer ces attaques restent restreintes et inefficaces.

Motivations

Les réseaux MANETs constituent, de part leur nature, un formidable challenge pour la sécurité informatique. Les spécificités de ces réseaux sont principalement : la transmission radio, la topologie dynamique, l'absence de toute infrastructure centrale, la nécessité de coopération des nœuds et l'hétérogénéité des nœuds avec pour certains des capacités restreintes. Toutes ces contraintes concourent à rendre la sécurité MANET difficile et complexe à appréhender.

L'authentification des nœuds et des messages échangés a constitué le point de départ pour les travaux de recherche sur la sécurité MANET, néanmoins d'autres besoins de sécurité on été soulevés en l'occurrence la base de confiance et la gestion et la distribution des clés, la sécurisation des protocoles de routage, la sécurisation de l'auto-configuration et la sécurité des données. Il existe de nombreux travaux théoriques mais peu d'applications pratiques qui puissent satisfaire l'ensemble des contraintes inhérentes aux réseaux MANETs.

Nos travaux de recherche s'intéressent de plus prés à cette thématique et constitue en soi un axe de recherche d'actualité et très prometteur pour des applications potentielles civiles et militaires. Nous nous sommes intéressés essentiellement à la conception de protocoles sécurisés et au développement d'algorithmes répondant aux besoins et exigences des réseaux MANETs, et qui concernent précisément l'initialisation d'un réseau MANET isolé et l'auto-configuration des nœuds joignant le réseau.

Contributions

Dans un premier temps, nous avons mis en lumière toutes les menaces de sécurité possibles ayant trait au service de configuration dynamique dans les réseaux mobiles ad hoc. Nous avons ensuite produit un état de l'art de toutes les approches proposées pour contrecarrer ces attaques. Notre propre analyse détaillée de ces propositions nous a révélé leurs faiblesses et limitations, ce qui nous a conduits à définir de nouvelles exigences impératives en matière de design des protocoles d'auto-configuration sécurisés pour les réseaux MANETs. Cette partie de travail a fait l'objet de l'article [33].

Nous avons proposé ensuite un nouveau protocole nommé TCSAP (Threshold Cryptography based Secure Autoconfiguration Protocol) pour sécuriser le service d'auto-configuration dans les réseaux MANETs. Le protocole fournit les tâches suivantes :

- le bootstrapping automatique sécurisé d'un réseau MANET spontané (isolé) grâce à un sous ensemble seuil de nœuds fondateurs exécutant une tâche d'auto-configuration coopérative avec les paramètres de sécurité et de réseaux nécessaires.
- la configuration automatique sécurisée d'un nouveau nœud joignant le réseau. Le service est assuré par un sous ensemble seuil de nœuds honnêtes déjà configurés.
- la prise en charge des problèmes liés au changement de topologie, notamment le partitionnement et la fusion des réseaux.

Notre solution est basée sur une authentification mutuelle, et un modèle de confiance d'Autorité de Certification et d'Auto-configuration complètement distribué, avec une réplication totale de l'information d'état. En utilisant le support de la cryptographie à seuil basée sur le problème du logarithme discret (DLP), et en déployant un nouveau concept de certificat conjointe liant l'adresse IP et la clé publique à l'identité d'un nœud, nous avons résolu irrévocablement le problème de quelques attaques comme l'attaque d'usurpation d'adresse IP et l'attaque Sybile, sans solution à ce jour par les mécanismes conventionnels.
Nous avons, par ailleurs, décrit tous les mécanismes de notre protocole TCSAP et développé les algorithmes associés. Une analyse de sécurité a été également élaborée démontrant la sécurité et la robustesse de notre protocole sous l'hypothèse d'un model d'adversaire fort.

L'implémentation du protocole TCSAP sous le simulateur NS2 en utilisant C++ et l'élaboration des scripts Otcl des simulations, nous ont permis d'évaluer les performances de notre solution et démontrer son efficacité. Ce travail s'est concrétisé par la publication de plusieurs articles [34-37].

Organisation de la thèse

Le premier chapitre présente une synthèse des schémas d'auto-configuration les plus pertinents pour les réseaux MANETs, suivie d'une comparaison qualitative de leurs performances.
Dans le second chapitre, nous mettrons en lumière les différentes menaces de sécurité possibles contre le service de configuration automatique dans les réseaux MANETs, ensuite nous analyserons les mécanismes de sécurité développés pour ce service et nous montrons leurs faiblesses. Enfin nous dégagerons les exigences impératives pour concevoir des schémas sécurisés et robustes permettant de contrecarrer toutes les attaques identifiées.
Le troisième chapitre est consacré aux modèles de confiance et d'authentification forte qui ont été proposés pour les réseaux MANETs. Nous montrerons que le modèle d'une infrastructure à clé publique distribuée présente les meilleurs atouts pour la solution que nous proposons. Nous y introduirons la cryptographie à seuil comme pilier nécessaire dans l'implémentation d'un tel modèle.
Dans le quatrième chapitre, nous développerons notre solution d'auto-initialisation et d'auto-configuration sécurisés pour les réseaux MANETs. Les spécifications et opérations du protocole TCSAP y seront détaillées.
Une preuve de concept par simulation et évaluation des performances du schéma proposé sera fournie dans le cinquième chapitre.
Enfin, nous conclurons ce travail et nous exposerons quelques perspectives.

CHAPITRE 1

L'auto-configuration dans les réseaux MANETS : Etat de l'art

I.1 Introduction

Le déploiement des réseaux mobiles ad hoc exige l'implémentation de mécanismes automatiques pour la configuration initiale des participants avec les paramètres de réseau et de sécurité nécessaires. En particulier, quand un nouveau nœud joint un MANET, il doit acquérir une adresse IP valide, c'est-à-dire topologiquement valable et globalement unique, avant de participer activement dans le réseau.
Ce problème fondamental et difficile a reçu une grande considération ces dernières années [7-13], [38-40] ; plusieurs solutions de configuration automatique ont été développées pour traiter les problèmes suivants :

- L'unicité d'adresse,
- L'initialisation de réseau,
- La fuite d'adresse,
- Le départ de nœud,
- Le partitionnement et la fusion de réseaux,
- L'efficacité,
- Le passage à l'échelle (scalability)

Les schémas d'auto-configuration représentent une partie importante des recherches dans le contexte MANET. Il existe une multitude d'approches, dont certaines mériteraient une étude approfondie. En 2005, l'IETF a créé le groupe de travail Autoconf [41] pour examiner les travaux liés à cette problématique et d'en dégager des standards. Malheureusement, jusqu'à présent tous les schémas proposés ont restés au stade de drafts et aucune norme n'a été ratifiée.

I.2 Contraintes des réseaux MANETs

Vu le caractère centralisé du service de configuration automatique DHCP [42], il ne peut être utilisé dans un contexte de réseaux MANETs qui sont par nature à topologie dynamique et sans infrastructure.
D'autre part, contrairement aux réseaux filaires, où un message à diffusion générale peut atteindre tous les nœuds sur un lien, les réseaux MANETs sont caractérisés par une topologie multi-sauts. Ainsi, même un message à diffusion générale devrait être conduit de saut en saut. C'est pourquoi même les mécanismes développés pour IPv6 [43] pour la configuration automatique en absence de serveurs DHCP ne pourraient pas s'appliquer directement.

Les contraintes d'énergie et de bande passante posent également des problèmes. Les nœuds dans un réseau MANET ont en général des réserves d'énergie limitées et doivent maintenir les communications de signalisation au minimum. La nature d'émission du support radio et les interférences entre les communications simultanées rendent les pertes de paquets relativement élevées, conduisant à des retransmissions de paquets et en conséquence à des consommations de puissance et de bande passante plus élevées, et ainsi qu'à des délais de communication plus importants.

La capacité en mémoire peut également jouer un rôle crucial dans le développement de protocoles d'auto-configuration pour les réseaux MANETs, en effet une information d'état sur les adresses déjà attribuées ou libres doit être maintenue d'une certaine façon pour garantir l'unicité et les meilleurs performances.

La mobilité (départ, arrivée ou migration) des nœuds (serveurs ou clients) dans un réseau MANET est une contrainte qui pèse beaucoup sur la gestion et la garantie du service d'auto-configuration.

I.3 Exigences des schémas d'auto-configuration

Compte tenu des contraintes citées précédemment, tout protocole d'auto-configuration optimal pour les réseaux MANETs doit satisfaire les objectives suivantes :

- Disponibilité du service quelque soit la topologie du réseau,
- Garantir en moyenne une latence d'auto-configuration raisonnable,

- Etre à complexité faible (overhead de calcul et de communication)
- Passage à l'échelle (Scalabilité),
- Prise en compte de la mobilité des nœuds et des changements de topologie (partitionnement/fusion),
- Assurer le service en cas de plusieurs demandes concurrentes.

I.4 Classification des approches d'auto-configuration MANETs proposées

Les techniques proposées pour l'auto-configuration dans les réseaux MANETs peuvent être classées en trois catégories selon la gestion de l'espace d'adressage : les solutions stateful (avec maintien de l'information d'état) et les solutions stateless (sans maintien de l'information d'état) et les solutions hybrides.

Dans la première catégorie, les adresses IP sont assignées par le réseau MANET, c'est-à-dire par un nœud ou par un ensemble de nœuds appartenant au réseau. Le réseau doit par conséquent maintenir des tables d'allocation contenant les adresses déjà attribuées ou libres. Pour ces approches, l'état de chaque adresse est tenu d'une telle manière que le réseau MANET ait une vision des adresses IP assignées et non assignées, ainsi la duplication d'adresse pourrait être évitée.

Contrairement dans la seconde catégorie, l'adresse IP est auto-assignée par le nœud joignant le réseau, le nœud choisit aléatoirement sa propre adresse et exécute un test de détection de duplication d'adresse afin de s'assurer de l'unicité de l'adresse choisie. La troisième catégorie combine les deux approches précédentes.

I.4.1 Approches Stateful

Toutes les approches stateful maintiennent des tables d'attribution d'adresse pour déterminer à chaque instant si une adresse est déjà assignée ou libre, ainsi les nœuds appartenant déjà au réseau MANET peuvent facilement assigner des adresses inutilisées aux nouveaux nœuds joignant le réseau. L'avantage de ces approches est la garantie de l'unicité de l'adresse assignée, néanmoins elles présentent le défi de synchroniser les tables d'attribution pour s'assurer que n'importe quelle adresse utilisée figure dans la table d'attribution.

Nous distinguons trois classes de schémas stateful selon que l'information d'état est reproduite totalement (full replication), reproduite partiellement (partial replication) ou n'est pas reproduite (null replication).

I.4.1.1 Approches stateful avec reproduction totale de l'information d'état

Dans cette classe, l'auto-configuration est basée sur une table d'adresse commune distribuée, chaque nœud peut assigner des adresses IP de cette table, en maintenant par conséquent les adresses déjà

assignées et les adresses en cours d'assignation. Ainsi, la synchronisation de ces tables distribuées constitue la tâche la plus critique et la plus complexe des protocoles de cette classe. Le plus connu de ces protocoles est MANETconf décrit ci après.

- **MANETconf**

Le protocole MANETconf proposé par S. Nesargi et R.Prakash [7] est basé sur le principe du DHCP distribué. La fonctionnalité du DHCP est répartie sur le réseau entier.
Dans MANETconf, chaque nœud a donc la possibilité d'assigner de nouvelles adresses puisqu'il maintient la table d'attribution. Quand un nouveau nœud (demandeur) joint le réseau, un de ses voisins (initiateur) pourrait choisir une adresse libre pour lui, le nouveau nœud diffuse localement (au premier saut) un message pour examiner son voisinage. Il choisit comme initiateur le premier voisin qui répond et entre en contact avec lui pour demander une adresse IP. L'initiateur choisit alors une adresse IP libre de sa table d'attribution et inonde le réseau entier pour avoir la permission d'assigner l'adresse choisie. Cette phase est exigée pour deux raisons : premièrement, les différentes tables ne peuvent être totalement synchronisées en raison du retard nécessaire de convergence de synchronisation, deuxièmement il est possible que deux nœuds choisissent simultanément la même adresse IP pour l'assigner à différents nœuds joignant le réseau.
Si tous les nœuds répondent positivement, il conclut que l'adresse choisie est libre et l'envoie au demandeur. Il fait ensuite l'inondation du réseau pour confirmer l'affectation de d'adresse en question et permettre à tous les nœuds de mettent à jour leurs tables. Si un ou plusieurs nœuds répondent négativement, l'initiateur conclut que l'adresse choisie est déjà assignée et répète un certain nombre de fois le procédé à partir du début. Si l'initiateur détecte qu'un ou plusieurs nœuds n'ont pas répondu, il reprend contact avec eux par des messages unicast demandant leurs permissions. Deux cas sont envisagés. Si le nœud concerné est encore présent dans le réseau, il répondra et l'initiateur pourrait alors continuer le processus de configuration. Si le nœud concerné est parti du réseau, l'initiateur ne recevra pas de réponse, ainsi il conclut après plusieurs tentatives que le nœud a quitté le réseau et inonde cette information pour informer le réseau entier au sujet de ce départ.

Les avantages de ce protocole sont qu'il garantit l'unicité de l'adresse assignée et qu'il est totalement distribué dans le sens où chaque nœud a la possibilité d'assigner de nouvelles adresses. Il fournit également un mécanisme pour prendre en charge la fusion de réseaux.

Chaque partition est identifiée par un identifiant (PartitionID). Cette identification se compose de la plus petite adresse dans le réseau et d'un UUID (Universal Unique ID) qui est fourni par le nœud possédant la plus petite adresse. L'identifiant (PartitionID) est inclus dans les messages périodiquement transmis. Un nœud détecte une fusion de partitions après la réception d'un message avec une identification de partition différente. Dans ce cas, les deux nœuds échangent leurs tables

d'attribution et les diffusent dans leurs partitions respectives. Chaque nœud qui trouve son adresse dans la table d'attribution de l'autre partition doit libérer cette adresse.

Les problèmes de MANETconf se résument dans les points suivants :

- complexité élevée en termes de communication, de maintien de table et synchronisation,
- délai important (tous les nœuds devraient donner leur permission à l'initiateur pour assigner une nouvelle adresse),
- sensibilité aux pertes en raison de sa dépendance sur des communications d'unicasts.

I.4.1.2 Approches stateful sans reproduction de l'information d'état

Dans cette classe, aucune reproduction des tables d'allocation d'adresses n'est faite. Chaque nœud détient sa propre table d'allocation, de sorte que la configuration d'un nouveau nœud se fait sans consultation des autres nœuds. Ces protocoles minimisent donc l'utilisation de l'espace mémoire.
Bien sur il existe plusieurs solutions appartenant à cette classe, nous présentons ici les plus remarquables dans la littérature.

a) Binary Split

L'approche Binary Split ou Buddy system, basée sur le principe de la division binaire est l'une des solutions d'auto-configuration stateful les plus intéressantes pour les réseaux MANETs. L'approche a été proposée la première fois par A. Misra et al. [44], puis améliorée respectivement par Mohsin et Prakash [9] et Thoppian et Prakash [45].
Dans ce protocole, l'espace d'adressage global est divisé en plusieurs blocs différents. Les tailles des blocs sont des puissances de deux, chaque division d'un bloc donne deux blocs de tailles égales. Chaque nœud est responsable d'un bloc et peut donc indépendamment assigner une nouvelle adresse IP et un nouveau pool (bloc d'adresses IP libres) sans consulter n'importe quel autre nœud dans le réseau.
On suppose dans cette solution qu'au début de la création du réseau MANET un seul nœud peut détenir la table entière des adresses.
Avant sa configuration, un nœud va envoyer périodiquement un message à diffusion générale réclamant une adresse IP. S'il ne reçoit aucune réponse, il s'auto-configure avec la première adresse IP de la plage d'adresses prédéfinie. S'il reçoit une ou plusieurs réponses, il (demandeur) choisit le premier qui répond (initiateur) et lui envoie une demande d'adresse, l'initiateur répond en divisant son propre pool d'adresse et renvoie la deuxième moitié avec une copie de la table d'adresse. Le demandeur s'auto-configure alors avec la première adresse IP dans le pool reçu et envoie un message de confirmation à son initiateur.

Si l'initiateur n'a aucune adresse disponible, la première solution est de demander à ses voisins un pool non vide en suivant une recherche en expansion d'anneau (incrémentation de sauts). Une autre solution consiste à rechercher le nœud détenant le plus grand pool, pour fournir la moitié de ce bloc au nœud qui joint le réseau.

Les nœuds synchronisent de temps en temps pour maintenir les adresses IP assignées et pour détecter toutes les fuites dans les pools disponibles. Chaque nœud maintient une table de correspondance entre les nœuds et les blocs d'adresses IP correspondants.

Les départs brusques de nœuds aussi bien que le partitionnement/fusion de réseau sont soutenus dans cette solution.

Le départ des nœuds peut avoir lieu d'une manière brusque ou annoncé, et pour chaque cas il y a une procédure à suivre. Si le départ est brusque, le nœud qui a agi en tant qu'initiateur dans sa configuration détectera son départ grâce à sa table de routage et il ajoutera alors le bloc du nœud sortant à son bloc libre d'adresse. Dans le cas du départ annoncé, le nœud qui part du réseau avertit ses voisins de son intention ; ainsi, le nœud qui a agi en tant qu'initiateur peut être contacté pour récupérer le bloc du nœud sortant.

Les plus grands avantages de ce protocole est que cela fonctionne bien pour le partitionnement/ fusion des réseaux, puisqu'il résous le problème de duplication d'adresses qui se pose dans ces cas. Pour distinguer différents réseaux, un réseau est associé toujours à une identification de réseau (PartitionID). L'identification de réseau est produite par le premier nœud qui lance le réseau.

Autre la garantie de l'unicité d'adresse, ce protocole totalement est distribué, chaque nœud est en mesure d'assigner de nouvelles adresses et l'affectation d'adresses dépend seulement de l'initiateur impliqué qui est un voisin du nœud demandeur, ainsi il est moins sensible aux pertes de réseau.
L'inconvénient principal de ce protocole est le mécanisme de synchronisation complexe exigeant l'inondation du réseau.
Thoppian et Prakash [45] ont proposé une amélioration du protocole Binary Split en considérant les aspects suivants : migration de demandeur, demandes concourantes d'adresse et pertes de message.

b) Prophet Allocation

L'idée derrière le schéma Prophet développé par H. Zhou et al. [10] est tel qu'au lieu de maintenir une table d'attribution et de travailler dur pour les synchroniser le long du réseau, chaque nœud maintient une fonction de génération stateful et une valeur d'état pour produire une séquence de nombres dans un

intervalle d'entiers R, qui sont utilisés comme adresses IP, ainsi l'attribution d'adresse est totalement décentralisée et produit un trafic nul. La difficulté dans ce protocole est de choisir la bonne fonction de génération. Une telle fonction devrait satisfaire les deux propriétés suivantes (si la plage d'adresses R est extrêmement grande) :

- L'intervalle entre deux occurrences du même nombre dans une séquence est extrêmement long.
- La probabilité que la fonction renvoie le même nombre pour deux valeurs différentes d'état est très faible.

La fonction de génération stateful produit des adresses IP telles que la probabilité d'un conflit d'adresses est très faible. Le premier nœud (Prophet) choisit aléatoirement une adresse IP et une première valeur d'état (seed). Ces valeurs sont employées comme entrées pour la fonction de génération stateful pour dériver la nouvelle adresse IP et la nouvelle valeur d'état pour n'importe quel nœud joignant le réseau.

Quand un nouveau nœud souhaite joindre le réseau, il envoie une émission locale à ses voisins. S'il ne reçoit aucune réponse, il conclut que c'est le seul nœud dans le réseau (Prophet) et se configure avec une adresse IP aléatoire et une valeur d'état (par défaut) pour la fonction prédéfinie de génération. Si le nouveau nœud reçoit plusieurs réponses de son voisinage, il contacte un de ses voisins pour demander une adresse IP. Le nœud initiateur emploie la fonction de génération pour obtenir une nouvelle adresse et une nouvelle valeur d'état et les fournit au nœud demandeur. Ensuite l'initiateur met à jour sa valeur d'état pour ne pas produire les mêmes nombres.

Dans [10], les auteurs proposent pour des réseaux de taille réaliste une fonction de génération f(n) basée sur un produit de nombres premiers avec chacun élevé à la puissance de la valeur d'état.

Le protocole est très simple pour être mis en application. Il ne produit presque aucun trafic supplémentaire. Néanmoins, on lui reproche les points suivants :

- il n'y a aucune preuve analytique que la fonction décrite remplit les conditions nécessaires.
- l'approche s'applique seulement pour de grands espaces d'adressage, et l'utilisation de l'espace d'adressage disponible n'est pas efficace.
- l'approche ne spécifie pas avec précision comment différents partitions de réseaux fusionnent.

- le protocole ne fait pas garantir l'unicité de l'adresse IP assigné même si la probabilité de la duplication d'adresse est très petite, et ne spécifie pas un mécanisme pour résoudre des conflits d'adresses au cas où ils se produiraient.

I.4.1.3 Approches stateful avec reproduction partielle de l'information d'état

Deux solutions existent dans cette classe, ils utilisent la technique « Binary Split » et la reproduction partielle de l'information d'état des adresses (blocs d'allocation), permettant ainsi de maintenir efficacement les tables d'allocation. La première proposé par L-H Chan et J-P Sheu [46] sauvegarde une copie des tables au niveau d'un seul nœud appelé Coor-Root ; dans la seconde les copies sont maintenues par un certain nombre de nœuds (Cluster-Heads), [8]. Les deux solutions se basent sur une approche hiérarchique où le service d'auto-configuration est partiellement distribué.

a) Approche Coor-Root

Dans cette proposition, quelques nœuds dans le réseau MANET sont définis comme Coordonnateurs et le premier qui initialise le réseau est appelés Coor-Root [46]. Les autres nœuds sont des nœuds ordinaires. Chaque coordonnateur est responsable d'assigner des adresses IP aux nouveaux nœuds de son voisinage à deux sauts et rapporte périodiquement le statut de sa table d'allocation au Coor-Root.

A l'initialisation du réseau MANET, le premier nœud coordonnateur détient le bloc global de l'espace d'adressage (pool), à l'arrivée d'un nouveau coordonnateur, le premier coordonnateur divisera son pool suivant le système Binary Split et distribuera la moitié inutilisée au nouveau coordonnateur. Le nouveau coordonnateur suit la même méthode pour réarranger son pool pour d'autres nouveaux coordonnateurs.

Le schéma proposé emploie des messages Hello, diffusés périodiquement pour échanger l'information sur les coordonnateurs. L'information échangée inclut l'identifiant du coordonnateur et la distance au coordonnateur en nombre de sauts. Chaque nœud ordinaire dans le réseau MANET enregistrera l'identification du coordonnateur le plus proche et sa distance.
Quand un nouveau nœud joint le réseau MANET, il devrait écouter ses voisins pour trouver le coordonnateur le plus proche grâce aux messages Hello envoyés par ses voisins. De cette façon, le nouveau nœud peut obtenir une adresse IP sans messages d'inondation dans le réseau.

Les coordonnateurs distribués à travers le réseau seront organisés pour former une topologie dynamique d'arbre appelée « Virtual C-tree ». La racine du C-tree est le premier coordonnateur dans

le réseau MANET, soit Coor-Root. Chaque nœud du C-tree est un coordonnateur. L'arbre « Virtual C-tree » est construit pour maintenir la conformité des pools d'adresses.

Avant qu'un nœud quitte le réseau, il doit libérer son adresse IP à son coordonnateur le plus proche. Puisque chaque nœud dans le réseau MANET a la connaissance de son coordonnateur le plus proche, un coordonnateur peut aisément récupérer les adresses IP libérées des nœuds partants. Pour réduire le trafic de signalisation dû à la récupération des adresses IP, l'arbre C-tree est utilisé pour échanger efficacement les messages de contrôle entre les coordonnateurs et pour maintenir dans la cohérence les pools d'adresses.

Le schéma [46] réduit l'utilisation de mémoire puisque seulement une partie de nœuds maintiennent les blocs disjoints d'adresses IP. Il fournit un délai réduit pour l'auto-configuration des nouveaux nœuds et une maintenance efficace des tables d'allocation. Cependant, il ne fournit pas des solutions pour l'emprunt d'adresse, ni de support pour le partitionnement/fusion de réseau, en plus l'information globale est maintenue par un seul nœud (Coor-Root).

b) Approche Quorum based

Ce schéma proposé par T. Xu et J. Wu [8] est une autre solution qui utilise la reproduction partielle de l'information d'état d'adresses. Il se base sur le clustering et le vote quorum. Le réseau MANET suit une organisation hiérarchique à deux niveaux, où plusieurs clusters s'étendant au maximum à deux sauts sont définis, chaque cluster est géré par un Cluster Head. Quand un nœud joint le réseau MANET, s'il trouve un Cluster Head au maximum à deux sauts, il obtient l'attribut d'un nœud ordinaire, sinon il devient un nouveau Cluster Head. Deux Cluster Heads ne peuvent être voisins.

Chaque Cluster Head maintient sa table d'allocation d'adresses (pool) obtenu par le système Binary Split. L'information d'état est reproduite localement au niveau des Cluster Heads adjacents se trouvant à trois sauts. Les Cluster Heads diffusent régulièrement des messages Hello dans un rayon à trois sauts pour garder la trace des Cluster Heads adjacents.
A la phase d'initialisation, le premier nœud dans le réseau MANET diffuse une requête de configuration avec une adresse libre. S'il n'obtient aucune réponse après une période prédéfinie, il répète le processus un certain nombre de fois avant de devenir le premier Cluster Head et obtient l'espace d'adressage tout entier pour l'allocation. Les nœuds qui arriveront ensuite devront intercepter les messages Hello diffusés par les Cluster Heads pour s'auto-configurer. Deux cas se présentent :

- premier cas : Si le nouveau nœud détecte un Cluster Head dans un rayon inferieur ou égal à deux sauts, il lui demande une adresse IP libre, le Cluster Head choisit une adresse et vérifie

si elle n'est pas déjà attribuée en demandant un vote quorum. En se référant à la figure 1.1, les nœuds en rouge sont des Cluster Heads et les nœuds en blanc sont des nœuds ordinaires. Le Cluster Head 1 ordonne des copies de son bloc d'adresse aux Cluster Heads adjacents à trois sauts : 2, 3, 4, 5, et 6 ; un exemple de vote quorum peut être {1,2,3,4}, {2,3,4,5}, {1,2,3,5}. Si l'adresse choisie est libre, alors le Cluster Head attribue cette adresse au nouveau nœud qui devient un nœud ordinaire. Le Cluster Head mis à jour l'information d'état dans le quorum.

- second cas : si le nouveau nœud ne détecte que des Cluster Heads à trois sauts, il dirige une demande de configuration vers l'un d'eux. A la réception de cette demande, le Cluster Head concerné assigne après un vote quorum la moitie de son pool au nouveau nœud qui devient un nouveau Cluster Head.

Le schéma proposé permet d'assurer une reproduction partielle de l'information d'état d'une manière plus ou moins distribuée. Il est hiérarchique donc scalable. Bien qu'il réduise le trafic de signalisation d'une manière satisfaisante, son mécanisme de vote de quorum où seulement une décision partielle est suffisante pour avoir une adresse IP valide n'assure pas l'uniformité de tables d'allocation. En plus le partitionnement et la fusion de réseaux sont mal supportés dans ce protocole.

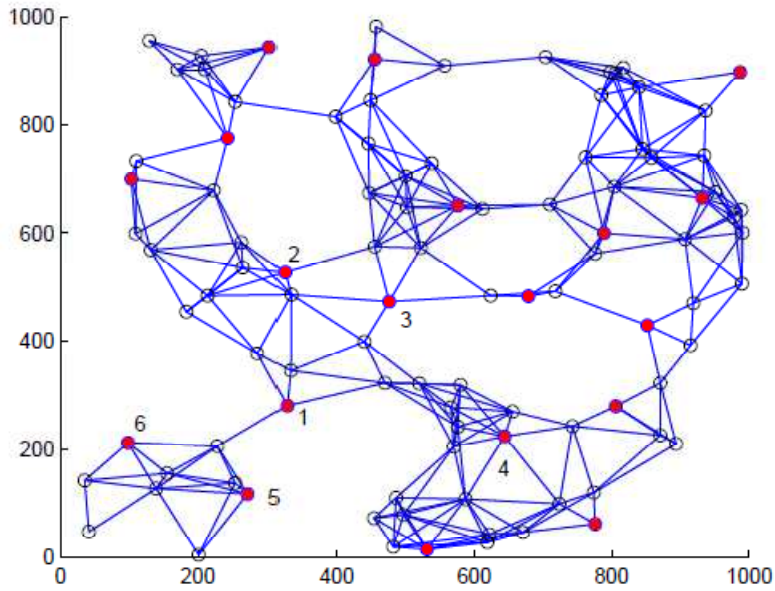


Figure 1.1 – Exemple de topologie à vote quorum

I.4.2 Approches Stateless

Toutes les approches stateless sont caractérisées par l'auto-assignation des adresses IP, qui signifie que chaque nœud choisit aléatoirement son adresse IP. Le nœud devrait exécuter un mécanisme de détection de duplication d'adresse (Duplicate Address Detection : DAD) pour s'assurer que son adresse IP choisie est unique dans le réseau. Le défi des approches stateless est de détecter dans un délai et un trafic modérés, la duplication potentielle d'adresse. Leur avantage réside dans leur simplicité relative comparée aux approches stateful.
Les deux protocoles stateless les plus pertinents sont présentés ici.

a) Approche Strong DAD

Parmi les travaux les plus remarquables dans des solutions stateful, nous notons la première approche Strong DAD proposée par C. Perkins et al. [39]. Nous pouvons considérer cette proposition comme une prolongation du protocole Zeroconf [47] pour les réseaux multi-sauts. Le mécanisme Strong DAD est la base pour toutes les approches stateless. Il se compose d'un mécanisme simple qui permet à un nœud ad hoc de choisir une adresse IP et d'examiner si elle est déjà attribuée ou non.

Spécifiquement, [39] décrit des mécanismes pour les réseaux IPv4 et IPv6, qui sont isolés de l'infrastructure Internet.

Un nœud du réseau MANET exécutant la configuration automatique sélectionne deux adresses, une adresse provisoire et une adresse tentative. L'adresse provisoire est employée seulement dans la phase d'initialisation comme adresse source pour des requêtes d'inondation utilisées pour détecter si l'adresse tentative est déjà employée ou non (processus DAD).

Le processus DAD consiste à diffuser sur le réseau, par le nouveau nœud, une requête de recherche de nœuds possédant la même adresse IP choisie. Si au cours d'une période de temps prédéfinie il ne reçoit rien, il répète la demande avec la même adresse tentative un nombre de fois spécifié pour s'assurer que cette adresse n'est pas utilisée avant qu'il libère l'adresse provisoire et adopte définitivement l'adresse tentative.

Si au cours de la période de temps prédéfinie, le nouveau nœud reçoit une réponse, il conclut que l'adresse tentative est déjà utilisée et recommence le processus en choisissant une autre adresse tentative.

Même si cette approche est la plus simple, nous pourrions imaginer qu'elle a beaucoup de problèmes et de limitations. La détection de duplication d'adresses effectuée est limitée à la phase d'initialisation. Ainsi, pour des raisons de perte temporaire de connectivité avec le réseau ou de pertes de messages le

processus auto-configuration peut mener à une duplication, le réseau ne peut pas résoudre cette duplication qui influe sur les performances du réseau. Les scénarios de partitionnement et de fusion de réseau ne sont pas considérés.

Ce protocole ne garantit pas l'unicité d'adresse, et la probabilité de duplication augmente avec la taille du réseau en cas d'espace d'adressage limité. Le protocole génère un trafic de signalisation important, chaque auto-configuration d'un nouveau nœud joignant le réseau engendre plusieurs inondations du réseau.

b) Approche Strong-Weak DAD

L'approche Strong-Weak DAD proposée par J. Jeong et al. [40] se base sur l'utilisation de deux mécanismes DAD. Le premier (Strong DAD) est exécuté dans la phase initiale de l'auto-configuration pour vérifier l'unicité de l'adresse, tandis que le second (Weak DAD) est proactivement réalisé pendant le processus de routage pour prendre en compte les duplications dues aux déconnections momentanés et la fusion des réseaux.
Le mécanisme Weak DAD proposé vise à prolonger le mécanisme de détection de duplication d'adresse pour la vie entière du réseau. Chaque nœud génère une clé à la phase d'initialisation, et la distribue avec son adresse IP dans tous les paquets de contrôle du protocole de routage ad hoc adopté. Cette clé sera utilisée pour détecter les adresses IP doubles.

Chaque nœud maintient dans sa table de routage les clés avec des adresses IP associées. Quand un nœud reçoit un message de routage avec une adresse IP qui existe déjà dans sa table de routage, il vérifie si les clés sont différentes. Si elles sont différentes, une adresse double est détectée et l'entrée est marquée non valide pendant qu'un message d'erreur est envoyé au nœud concerné afin de lui indiquer qu'une duplication d'adresse s'est produite. Le nœud en question devrait s'auto-configurer avec une nouvelle adresse IP en utilisant le mécanisme Strong DAD.

Le processus Weak DAD doit soutenir l'identification d'un nœud au moyen d'une paire « clé-adresse IP » et dépend complètement du protocole de routage. Cela fonctionnera seulement avec un protocole proactif qui met à jour les routes constamment, mais avec un protocole réactif il y aura des nœuds qui pourraient ne jamais détecter la duplication des adresses IP.

L'inconvénient principal du processus Weak DAD est donc sa dépendance du protocole de routage. Il exige quelques changements à la couche réseau pour soutenir l'introduction de l'identification par la paire« clé-adresse IP ».

Comme nous l'avons signalé plus haut, le mécanisme Strong DAD exige des inondations de messages pour découvrir s'il y a un conflit de duplication ou non. Le processus Weak DAD n'exige aucun trafic additionnel pour le mécanisme d'auto-configuration, mais le prix est le trafic provoqué par l'intégration de la clé dans les paquets de routages.

I.4.3 Solutions Hybrides

Les solutions hybrides combinent les fonctionnalités des approches stateful et sateless ; d'une part une certaine information d'état des adresses attribuées est maintenue, d'autre part les nœuds s'auto-configurent en self service et des mécanismes DAD sont exécutés pour résoudre les conflits d'adresses.

a) Approche PACMAN (Passive DAD)

L'approche développée par K. Weniger, appelé PACMAN (Passive Auto-configuration for Mobile Ad hoc Networks) [12] est une solution hybride qui emploie des éléments des protocoles stateful et stateless. Elle propose une affectation d'adresses utilisant un algorithme probabiliste, et une détection passive de duplication d'adresse basée sur des anomalies dans le trafic du protocole de routage.

Chaque nœud déjà configuré doit maintenir une table d'allocation des adresses déjà utilisées. Ces tables sont maintenues en utilisant les informations du trafic de routage.

Un nœud exécutant le protocole PACMAN s'auto-configure avec une adresse IP en utilisant un algorithme probabiliste. Basé sur une probabilité de conflit prédéfinie $P_{conflit}$, une estimation du nombre de nœuds et la table d'allocation d'adresse fournie par un voisin, l'algorithme calcule la taille de l'espace d'adressage virtuel, choisit aléatoirement une adresse à partir de cet espace et s'assure que l'adresse n'a pas été déjà assignée grâce à la table locale d'attribution.

La probabilité de conflit estimée en utilisant le paradoxe des anniversaires, est donnée par l'équation (I.1) [12], où r est la taille de l'espace d'adressage, n le nombre de nœuds dans le réseau MANET et j le nombre d'adresses déjà attribuées et non connues.

$$P_{conflit} \approx 1 - (r.e)^{-j} \frac{(r-n+j)^{r-n+j+\frac{1}{2}}}{(r-n)^{r-n+\frac{1}{2}}} \qquad \textbf{(I.1)}$$

L'adresse choisie est assignée immédiatement. Ainsi, le temps d'auto-configuration d'un nouveau nœud est pratiquement réduit, mais l'adresse peut être dupliquée avec une certaine probabilité. Compte tenu de la table d'allocation, cette probabilité est habituellement zéro. Seulement si un nombre important de nœuds joignent le réseau simultanément, la table d'attribution n'est pas mise à jour et des

conflits peuvent se produire. Cependant, ils sont détectés et résolu par le mécanisme DAD passif (Passive DAD) [48].

Dans le fonctionnement de ce mécanisme, un nœud analyse les paquets reçus du protocole de routage pour dériver des indications au sujet de conflits d'adresses. L'idée fondamentale est d'appliquer divers analyses aux paquets reçus du protocole de routage, qui exploitent les événements du protocole qui ne se produisent jamais en cas d'adresse unique, mais toujours dans le cas d'une adresse double, ou qui se produire rarement en cas d'adresse unique, mais souvent en cas d'adresse double.
Dans le premier cas, un conflit est sûr si l'événement se produit une fois, tandis que dans le deuxième cas, un conflit est présent seulement avec une certaine probabilité. Dans ce cas-ci, la surveillance à long terme peut être nécessaire. De tels algorithmes sont probabilistes. Chaque nœud maintient une table avec la probabilité de conflit pour chaque adresse. Chaque fois que l'algorithme (Passive DAD) identifie une indication au sujet d'un conflit, il modifie la probabilité pour l'adresse correspondante dans la table. Si un certain seuil est atteint, l'algorithme (Passive DAD) suppose que l'adresse est en effet double et déclenche la résolution de conflit.
L'avantage de ce protocole est qu'aucun trafic de signalisation additionnel n'est produit. Cependant, il exige l'analyse complexe d'information de routage et s'applique plutôt pour des protocoles de routage proactifs.

b) Approches Leader Based

La classe des approches Leader based (K. Weniger et M. Zitterbart [49] et Y. Sun et E.M. Belding-Royer [13]) est également intéressante. Au lieu que tous les nœuds doivent participer au processus de DAD, la responsabilité est donnée à certains nœuds spéciaux (Leaders [49] ou Address Authority [13]). L'approche leader based a été proposée pour résoudre le problème des nœuds temporellement déconnectés du réseau, et qui ne peuvent pas envoyer des messages de réponse aux requêtes DAD.
L'information d'état d'attribution d'adresses doit être maintenue auprès des Leaders qui annoncent périodiquement l'information nécessaire pour permettre à un nouveau nœud d'être configuré, pour assurer l'unicité d'adresse et pour le support du partitionnement et la fusion du réseau.

La solution décrite dans [49] prolonge les mécanismes du protocole de découverte de voisins (Neighbor Discovery) et les mécanismes d'auto-configuration stateless du protocole IPv6 pour fonctionner dans un contexte MANET.

Pour permettre la configuration d'adresses uniques, une hiérarchie est établie par les nœuds Leaders qui configurent un groupe de nœuds en publiant des annonces de routeur (Router Advertisement, RA) dans un rayon prédéfini (scope), ces annonces contiennent le préfixe de sous-réseau qui est unique

pour chaque Leader, et qui est choisi aléatoirement par le Leader. Ainsi la détection de duplication d'adresses (DAD) doit être effectuée dans le réseau MANET entier uniquement entre les Leaders.

Les nouveaux nœuds joignant le réseau MANET, peuvent s'auto-configurer grâce aux annonces publiés par les Leaders dans leurs scopes, en choisissant aléatoirement une adresse tentative. Un test DAD est effectué par le nouveau nœud dans le scope du Leader. Si le test DAD réussit, l'adresse peut être considérée en tant que valable pour une certaine période. La solution proposée utilise un DAD périodique pour assurer l'unicité d'adresse dans le scope du Leader.

Le protocole s'appui sur une approche hiérarchique ce qui lui permet d'être scalable. Néanmoins, il nécessite un algorithme pour l'élection de Leaders, et produit un trafic de signalisation toujours élevé dû à l'inondation du réseau pour le cas du DAD des Leaders, et au niveau de chaque scope de Leader pour le cas du DAD des nœuds s'auto-configurant.

Dans [13], les auteurs proposent un protocole nommé DACP (Dynamic Address Configuration Protocol). Pour assurer la détection d'adresse double tout en réduisant au minimum la participation des nœuds de réseau, une autorité d'adresse AA (Address Authority) est élue pour maintenir l'information d'état du réseau, tel que les adresses des nœuds, leurs durées de vie et l'identifiant unique du réseau.

Cependant, cette approche ne se fonde pas sur la seule autorité élue AA pour assigner des adresses. Au lieu de cela, l'affectation d'adresses est accomplie d'une façon distribuée. À la phase d'une nouvelle auto-configuration, un nœud choisit deux adresses, une adresse provisoire et une adresse tentative, et exécute le mécanisme Strong DAD. Si la configuration automatique d'adresse est réussie, le nouveau nœud doit enregistrer son adresse tentative auprès de l'autorité AA. Par conséquent il doit attendre une annonce (Advertisement) en provenance de l'autorité AA. A la réception de cette annonce, le nouveau nœud lance une demande d'enregistrement et attend la confirmation de l'autorité AA. Seulement après la confirmation, le nœud peut commencer à employer cette adresse. Après un enregistrement réussi, le nœud déclenche un temporisateur et réinitialise le procédé d'enregistrement chaque fois que le temporisateur expire.

En plus de tenir les états de toutes les adresses IP assignées, l'autorité AA peut aider à détecter la duplication d'adresse dans la phase d'initialisation par la réponse au Strong DAD destinée à une adresse tentative déjà utilisée. Ceci est important particulièrement quand le nœud intéressé est déconnecté temporairement.
A l'initialisation du réseau MANET, le premier nœud qui obtient une adresse IP unique devient l'autorité d'adresse AA dans le réseau. L'autorité choisit un identifiant unique pour le réseau, et

l'annonce périodiquement pour identifier le réseau. Si un nœud n'entend aucune annonce de la part de l'autorité pendant une certaine période, il considère qu'il y a eu un partitionnement du réseau, il devient la nouvelle AA, et produit un nouveau identifiant de réseau. Quand un nœud détecte un nouveau identifiant de réseau, il doit enregistrer son adresse auprès de la nouvelle AA, ainsi aucun changement d'adresse n'est nécessaire. La fusion de réseau est détectée par la présence de deux identifiants de réseau. Dans ce cas-ci, seules les autorités sont impliquées dans la détection des conflits d'adresses en échangeant leurs différentes tables.

Ce protocole ajoute la robustesse au mécanisme Strong DAD, en garantissant la détection de duplication. En même temps, il propose un mécanisme efficace pour détecter et manipuler le partitionnement et la fusion du réseau.

Le protocole a toutefois quelques problèmes. D'abord, les trafics de signalisation produits par la détection de duplication et les inondations périodiques de l'AA sont très élevés. En second lieu, la configuration automatique d'adresse dépend d'une entité centrale qui exige de tous les nœuds d'enregistrer leurs adresses par l'unicast. Ce mécanisme augmente le délai de l'auto-configuration.

I.5 Comparaison qualitative des approches d'auto-configuration

Dans ce paragraphe, nous donnons une récapitulation des différentes approches d'auto-configuration décrites dans ce chapitre, avec une comparaison des caractéristiques permettant d'évaluer qualitativement leurs performances (Tableau 1.1).

		Protocole	Caractéristiques					
			Garantie de l'unicité	Délai de configuration	Trafic de Signalisation	Partitionnement Fusion	Scalabilité	Dépendance du protocole de routage
Stateful	Full Replication	MANETconf	oui	élevé	élevé	oui	non	non
	Partial Replication	Quorum Based	oui	moyen	moyen	oui	oui	non
	Null Replication	Binary Split	oui	moyen	moyen	oui	oui	non
		Prophet	non	faible	faible	non	non	non
Stateless		Strong DAD	non	élevé	élevé	non	non	non
		Strong-Weak DAD	non	élevé	élevé	non	non	non
Hybride		PACMAN	non	élevé	élevé	non	non	oui
	Leader Based	DACP	non	moyen	élevé	non	oui	non
		Weniger	non	moyen	moyen	non	oui	non

Tableau 1.1 – Comparaison qualitative des performances des approches d'auto-configuration pour les réseaux MANETs

I.6 Conclusion

Etant donné qu'il n'est pas possible d'établir une liste exhaustive des solutions d'auto-configuration proposées pour les réseaux MANETs car celles-ci sont déjà nombreuses, nous avons présenté dans ce chapitre les plus pertinentes. En vertu des exigences imposées par les contraintes MANETs et les besoins du service d'auto-configuration, chacune des solutions proposées a ses avantages et ses inconvénients. Toutefois, elles souffrent toutes d'un problème commun : absence de la sécurité.

Beaucoup moins d'attention a été accordée à la sécurité dans la conception de tels protocoles. Ainsi, plusieurs vulnérabilités des réseaux MANETs peuvent être exploitées par un adversaire pour effectuer des attaques aussi actives que passives, altérant en conséquence la disponibilité du service et le comportement du réseau entier.

Le chapitre suivant met en lumière cette problématique et donne quelques exigences et recommandations pour la conception de protocoles d'auto-configuration sécurisés.

CHAPITRE 2

Sécurité de l'auto-configuration des réseaux MANETs: Menaces réelles et exigences

II.1 Introduction

Dans le cadre de la sécurité des réseaux MANETs, la majeure partie des recherches a été concentrée principalement sur les modèles de confiance et d'authentification, sur la sécurité des protocoles de routage, et beaucoup moins d'attention a été accordée au problème de sécurité du service de l'auto-configuration. Cependant, le manque de sécurité dans la conception des protocoles pour un tel service, ouvre la possibilité de menaces réelles dues aux vulnérabilités bien connues et qui sont spécifiques au paradigme ad hoc. Ceci peut mener par conséquent à des attaques sérieuses dans des environnements potentiellement hostiles.

Quelques travaux [27-32] ont tenté de résoudre ce problème, mais aucun d'eux n'a apporté de solution satisfaisante. Le problème de sécurité du service d'auto-configuration dans les réseaux MANETs reste donc, à ce jour, encore ouvert. Le but de ce chapitre est d'identifier toutes les menaces possibles et de définir, à travers une analyse détaillée des propositions précédentes, les exigences impératives dans la conception de tels schémas.

II.2 Menaces de sécurité

Dans tous les schémas d'auto-configuration proposés pour les réseaux MANETs, le fonctionnement correct du service dépend du comportement correct des nœuds participants. Cependant, les nœuds ne sont pas en réalité tous honnêtes et des nœuds malveillants ou compromis peuvent être aussi présents dans le réseau, posant potentiellement une série de problèmes possibles.

A travers l'analyse des protocoles d'auto-configuration existants pour les réseaux MANETs, nous avons identifié plusieurs attaques [33], qui sont décrites ci-après. Le tableau II.1 résume les failles de quelques protocoles vis-à-vis ces attaques.

II.2.1 Attaque d'usurpation d'adresse IP (IP Spoofing)

Un nœud malicieux pourrait usurper une adresse IP déjà assigné ou libre. Dans le premier cas, il usurpe n'importe quel nœud configuré et détourne son trafic ; dans le second cas, il s'auto-assigne une adresse IP valide et globalement libre pour participer légitiment au réseau, et conduire des attaques actives ou simplement causer du déni de service (Denial of Service, DoS).

II.2.2 Attaque d'épuisement de l'espace d'adressage (Exhaustion Address Space)

Un nœud malicieux pourrait réclamer le maximum d'adresses IP dans le but d'épuiser l'espace d'adressage. Il pourrait également demander l'attribution d'adresse pour des nœuds fantômes c'est-à-dire n'ayant aucune existence dans le réseau. En acquérant toutes les adresses valides, un nœud malicieux peut priver beaucoup d'autres nœuds à s'auto-configurer.

II.2.3 Attaque de conflit d'adresses (Conflict Address)

En jouant le rôle d'un initiateur, un nœud malicieux pourrait assigner à un demandeur une adresse dupliquée avec probablement un pool d'adresses déjà utilisées. Il pourrait effectuer, dans le processus du DAD, une attaque de trou noir pour des messages de réponse. Ceci peut mener à des conflits d'adresses sérieuses dans le réseau.

II.2.4 Attaque de faux conflit d'adresses (False Conflict Address)

Un nœud malicieux pourrait répondre, dans le processus de DAD, par un message de conflit d'adresses réclamant que l'adresse IP tentative est déjà assignée, alors qu'elle ne l'est pas réellement; il peut au besoin changer temporairement son adresse IP pour effectuer cette attaque.

II.2.5 Attaque sybile

L'attaque sybile est une attaque particulièrement nocive dans un réseau MANET où un nœud malicieux réclame d'une manière illégitime des identités multiples. Un nœud sybile peut fabriquer une nouvelle identité ou voler une identité d'un nœud légitime. L'obtention de plusieurs adresses IP par un nœud sybile facilite beaucoup son attaque. C'est en effet possible avec tous les mécanismes d'auto-configuration décrits dans le chapitre 1.

II.2.6 Attaque de DoS (surcharge du réseau)

Dans un schéma demandeur-initiateur, un nœud malicieux pourrait agir en tant que demandeur et envoyer simultanément des messages de demande d'adresse à un nombre important d'initiateurs. Un nœud malicieux pourrait également envoyer beaucoup de messages DAD pour différentes adresses IP tentatives, conduisant à la surcharge du réseau.

	Attaque IP Spoofing	Attaque d'épuisement de l'espace d'adressage	Attaque de conflit d'adresses	Attaque de faux conflit d'adresses	Attaque sybile	Attaque de DoS (surcharge du réseau)
Binary Split	OUI	OUI	OUI	NON	OUI	NON
MANETconf	OUI	OUI	OUI	OUI	OUI	OUI
Prophet	OUI	OUI	OUI	NON	OUI	NON
Strong DAD	OUI	OUI	OUI	OUI	OUI	OUI
Strong-Weak DAD	OUI	OUI	OUI	OUI	OUI	OUI
PACMAN	OUI	OUI	OUI	NON	OUI	NON
Leader Based	OUI	OUI	OUI	OUI	OUI	NON
Coor-Root	OUI	OUI	OUI	NON	OUI	NON
Quorum based	OUI	OUI	OUI	NON	OUI	NON

Tableau 2.1 – Attaques possibles sur quelques schémas d'auto-configuration pour les réseaux MANETs

II.3 Aperçu et analyse des schémas de sécurité précédents

Au meilleur de notre connaissance, [27-32] sont les seuls papiers existants ayant traité le problème de sécurité de l'auto-configuration dans les réseaux MANETs. Un examen court de ces documents est présenté dans cette section, suivie d'une analyse mettant en évidence les limites des solutions proposées.

II.3.1 Autorité de certification on-line à un saut

En premier lieu, F. Buiati et al. [27] ont présenté une extension de sécurité au protocole DCDP (Dynamic Configuration and Distribution Protocol), également connu par le concept de buddy system ou binary split [9,44]. Leur proposition est basée sur l'imposition aux nouveaux nœuds désirant joindre le réseau MANET d'obtenir un certificat on-line avant de postuler pour une configuration automatique d'adresse IP. Ce certificat lie l'identité d'un nœud à sa clé publique, il est délivré au premier saut par une autorité de certification distribuée basée sur le modèle de confiance à seuil (***k***, ***n***); le certificat est signé par au moins ***k*** nœuds voisins à un saut parmi n nœuds participants dans le réseau entier. Dans le processus d'auto-configuration, tous les messages de contrôle doivent être authentifiés avec non-répudiation. Ceci est réalisé par une extension MAE (MANET Authentication Extension) [50] ajoutée à chaque message.

Le problème principal de la solution proposée est que la disponibilité du service de certification n'est pas garantie. Pour un nouveau nœud joignant le réseau, trouver au moins ***k*** nœuds voisins à un saut n'est pas toujours possible. C'est l'un des inconvénients de cette solution.

Pour assurer un fonctionnement correct du service d'auto-configuration, les auteurs ont proposé d'utiliser deux mécanismes : détection des actions malveillantes contre le service d'auto-configuration, et génération d'accusations contre les nœuds malicieux. Mais ils n'ont donné aucun détail quant à ces mécanismes. Selon [9], pour le maintient des adresses IP attribuées chaque nœud dans le réseau MANET doit garder une table de correspondance entre les identités et les adresses IP assignées. Il n'est pas montré comment construire d'une manière sécurisée ces tables. En conséquence, la solution proposée ne permet pas d'empêcher les attaques d'épuisement de l'espace d'adressage et de DoS.

Les auteurs n'ont pas traité également dans leur solution l'attaque d'usurpation d'adresse IP. Un nœud malicieux avec un certificat valide peut facilement usurper des adresses IP déjà utilisées pour détourner le trafic ou pour exécuter certaines attaques comme l'attaque de conflit d'adresses ou l'attaque sybile. A noter que l'attaque d'usurpation d'adresse IP peut être empêchée si les tables de correspondance ci-dessus sont maintenues d'une manière sécurisée.

II.3.2 Pré-authentification et autorisations à seuil

A. Cavalli et la JM Orset ont proposé dans [28] une autre solution de sécurité appliquée au schéma d'auto-configuration binary split [9]. Un nouveau nœud demandant une adresse IP et un pool d'adresses libres doit tout d'abord collecter un certain nombre seuil d'autorisations fournies par les nœuds voisins à un saut. Chaque autorisation est accordée après une authentification mutuelle entre le nœud demandeur et un nœud voisin faisant déjà parti du réseau. Nous avons identifié quatre problèmes avec cette solution :

- Un nouveau nœud joignant le réseau ne peut pas toujours avoir un nombre suffisant (au moins égal au seuil) de nœuds voisins à un saut.

- L'adresse IP et le pool d'adresses libres sont attribués par un seul nœud voisin qui prétend tenir le grand pool. Ce nœud peut être un nœud malicieux, effectuant l'attaque de conflit d'adresses.

- Le demandeur peut également être malicieux, la solution proposée par les auteurs n'est pas protégée contre l'attaque d'épuisement de l'espace d'adressage (considérer par exemple un modèle d'adversaire mobile [27]).

- Les auteurs n'ont adressé aucun mécanisme pour contrecarrer l'attaque d'usurpation d'adresse IP. En effet, un nœud malicieux pourrait à tout moment usurper n'importe quelle adresse IP valide pour prendre une partie active dans le réseau ou pour détourner juste le trafic, puisqu'il n'y a aucun mécanisme permettant de vérifier le lien de l'adresse IP avec l'identité du nœud.

II.3.3 Adresse IP basée sur le haché de la clé publique

P. Wang et al. [29] ont décrit un schéma original d'auto-authentification pour l'auto-configuration d'adresse IP, qui lie l'adresse IP d'un nœud avec sa clé publique par l'intermédiaire d'une fonction de hachage à sens unique. L'adresse IP d'un nœud est obtenue près hachage de sa clé publique. Le schéma emploie également le processus DAD pour vérifier l'unicité de l'adresse IP. Pour résoudre le problème des collisions, les auteurs ont proposé de recalculer le haché avec de nouvelles paires de clés publiques/privées. La solution proposée présente également quelques inconvénients :

- Le schéma proposé est stateless, par conséquent n'importe quel nœud peut tout seul produire une adresse IP pour participer au réseau.

- Tenant compte du problème de collision, les clés publiques sont produites autant que c'est nécessaire, et ils ne sont signés par aucune autorité de certification. Ceci donne la possibilité à un nœud malicieux d'avoir plusieurs clés publiques et plusieurs adresses IP; ainsi il peut facilement effectuer l'attaque sybile.

- Un nœud malicieux pourrait produire à l'avance aléatoirement plusieurs paires de clés publiques et d'adresses IP associées et les sauvegarder dans une table. Puis, il effectuerait des attaques de faux conflit d'adresses. En recevant une demande DAD, il vérifie juste si l'adresse

IP (adresse cible) dans le message reçu correspond à n'importe quelle adresse de sa table, puis il effectue l'attaque en changeant son adresse IP actuelle à l'adresse cible.

- Les auteurs précisent la nécessité de l'authentification des nœuds pour parer l'attaque d'usurpation d'adresse IP, mais leur solution ne traite pas cette question. Il nous parait indispensable que chaque nouvelle clé publique produite soit signée par une autorité de certification on-line. Néanmoins, nous nous interrogeons comment cela peut être effectué dans le schéma proposé.

II.3.4 Modèle de sécurité basé sur une valeur de confiance

S. HU et C.J. Mitchell [30] ont proposé, pour améliorer la sécurité des schémas d'auto-configuration de type demandeur-initiateur , d'utiliser un modèle de sécurité basé sur une valeur de confiance. Dans ce modèle, chaque nœud i maintient une valeur de confiance seuil T_i*, et considère un nœud j digne de confiance si et seulement si la valeur de confiance $T_i(j)$ maintenue par le nœud i pour le nœud j est telle que : $T_i(j) \geq T_i$*, autrement il le considère en tant que malicieux.
Chaque nœud maintient également une liste noire contenant les identités des nœuds malicieux. Cette liste noire est dynamiquement mise à jour en utilisant des informations recueillies du réseau. Pour choisir un nœud digne de confiance (soit i) comme initiateur, les auteurs proposent de combiner des valeurs de confiance maintenues par les nœuds voisins du nœud i. Pour vérifier l'unicité de l'adresse IP, le protocole DAD est exécuté. Pour sécuriser ce processus, l'initiateur emploie la valeur de confiance qu'il maintient/calcule pour le nœud répondeur.
Pour traiter certaines vulnérabilités dans leur modèle, les auteurs font beaucoup d'hypothèses qui ne sont pas réalistes dans un environnement hostile, ce qui affaiblit leur solution et la rend peu adaptée pour les réseaux MANETs. En effet, nous avons soulevé des inconvénients et des limites de sécurité que nous résumons dans les points suivants :

- Pour mesurer la confiance, chaque nœud doit calculer des valeurs de confiance de ses nœuds voisins à un saut ou d'autres nœuds à plusieurs sauts. Dans le premier cas, le calcul est basé sur des informations recueillies par observation passive par le nœud au sujet du comportement de ses nœuds voisins. Cette approche n'est pas efficace vu qu'elle est basée sur l'analyse du trafic. Les auteurs ajoutent également que des problèmes potentiels pourraient surgir dans ce processus concernant la densité des nœuds et les protocoles de la couche MAC employés. Dans le deuxième cas, pour déterminer la valeur de confiance pour un nœud distant de plusieurs sauts, les auteurs proposent de modifier la méthode Route Discovery du protocole de routage DSR [2] en y ajoutant une liste contenant la valeur de confiance pour chaque nœud marqué dans la route. Chaque nœud doit apposer une valeur de confiance pour son successeur.

Cette approche est vulnérable à l'attaque d'intégrité. Un nœud malveillant peut intentionnellement changer la liste des valeurs de confiance. Même s'il sera facile d'apprendre qu'il y a au moins un nœud malveillant dans l'itinéraire, il n'est pas possible de l'identifier. Ceci forcera l'initiateur à répéter le processus DAD; par son action, le nœud malicieux exécute en fait une attaque de DoS qui aura pour conséquence une augmentation de la latence.

- Les auteurs prétendent que la probabilité qu'un nœud malveillant sera choisi comme initiateur est basse, sous l'hypothèse qu'une majorité des nœuds dans le réseau soient honnêtes. Cette prétention n'est pas fondée. En effet, considérer le cas d'un ensemble de nœuds voisins dans lequel la majorité des nœuds sont malicieux, la probabilité ci-dessus dépend ainsi de la distribution des nœuds malicieux même s'ils sont minoritaires dans le réseau.

- Pour causer une attaque de DoS, un nœud malicieux pourrait agir en tant que demandeur et envoyer des messages de demande d'adresse à plusieurs initiateurs simultanément. L'approche décrite précédemment ne peut pas empêcher ce type d'attaque.

- Comme précisé par les auteurs eux-mêmes, le modèle proposé souffre des attaques d'usurpation d'adresse IP et sybile, et il est également vulnérable aux nœuds malveillants qui se comportent avec malveillance seulement en ce qui concerne la fonctionnalité du modèle de confiance proposé.

II.3.5 Signature des messages de l'autorité d'adresse

Le travail de Langer et Kühnert [31] vise à sécuriser le protocole ODACP (Optimized Dynamic Address Configuration Protocol), [51]. Le protocole ODACP emploie le mécanisme DAD combiné avec un schéma leader based (voir chapitre 1, section 4). Dans [51], une autorité AA (Adress Authority) doit être élue par les nœuds participants dans le réseau MANET.

La solution décrite dans [31] est principalement basée sur deux mécanismes. D'abord, pour empêcher un nœud malveillant d'envoyer des messages en prétendant leur origine l'autorité AA, ou de modifier et d'envoyer à un nœud demandeur des messages issus de l'autorité AA, un cheksum chiffré est ajouté par l'autorité AA à tout message de réponse à la requête DAD. Le protocole d'échange de clés Diffie-Hellman est utilisé. Ensuite, pour empêcher le comportement malveillant du demandeur (attaque d'épuisement de l'espace d'adressage), les auteurs ont introduit comme paramètres additionnels dans la table d'allocation d'adresses de l'autorité AA, un compteur de demandeur (requester counter) et un horodateur (timestamp) donnant l'instant de la dernière demande. Ils réclament une valeur seuil égal à

6 pour le compteur afin de permettre à un nœud de réessayer sa configuration automatique en cas d'erreurs ou de problèmes de communication.

La solution proposée [31] présente quelques faiblesses :

- Aucun modèle de confiance n'a été défini par les auteurs qui n'assument ainsi aucune authentification des messages ni dans le processus DAD ni dans l'enregistrement d'adresse. Ceci peut mener à plusieurs attaques (usurpation d'adresse IP, épuisement de l'espace d'adressage). Quand un nœud obtient une adresse IP par le processus DAD, il unicast une demande d'enregistrement à AA comprenant son adresse IP, son adresse MAC et une durée de vie pour l'adresse IP acquise. Même si les auteurs ont ajouté un compteur de demandeur au niveau de la table d'allocation de l'autorité AA, il est évidemment insuffisant d'empêcher un nœud malicieux d'exécuter une attaque d'épuisement de l'espace d'adressage, puisque ni la demande d'adresse ni l'enregistrement d'adresse n'est signé. Le nœud malicieux pourrait simplement transmettre une fausse adresse MAC. Il pourrait également effectuer une attaque d'usurpation d'adresse MAC en retransmettant des demandes d'adresses ou d'enregistrement au nom des victimes visées à déborder leur compteurs.

- Puisqu'il n'y a aucune authentification mutuelle, il est bien connu que l'attaque man-in-the middle est facilement exécutée dans le protocole d'échange de clés Diffie-Hellman que les auteurs ont adopté dans leur solution. L'environnement sans fil ouvert et les communications multi-hops facilitent énormément cette attaque.

- L'autorité AA centralisée simplifie l'administration du service de configuration automatique, mais elle constitue une faiblesse de sécurité, particulièrement quand des messages d'annonce issus de multiples AA se produisent dans un réseau après fusion. Ce problème a été traité par les auteurs qui ont proposé une solution pour empêcher un nœud malveillant de devenir, dans une situation de fusion particulièrement, la nouvelle AA pour le réseau entier. La proposition emploie un procédé de validation entre les AA des réseaux fusionnant. Leur solution est trop complexe empêchant le protocole de fonctionner rapidement et détecter correctement des fusions. Les problèmes de sécurité dans ce cas-ci et dans l'initialisation de réseau sont laissés ouverts.

II.3.6 Secure Prophet

Dans [32], H. Zhou a proposé un mécanisme pour garantir l'unicité des adresses construites par le schéma d'allocation d'adresse prophet [10]. L'auteur a mis en évidence plusieurs attaques (IP

spoofing, State pollution, sybile) pouvant altérer le fonctionnement correct du protocole prophet. Il a proposé en conséquence, une extension des messages réponse de l'initiateur avec des paramètres additionnels permettant au demandeur de calculer de nouvelles adresses jamais attribuées. L'auteur prétend que son mécanisme est capable de prévenir les attaques identifiées dans prophet et de fournir la garantie de l'unicité dans un environnement non sécurisé.

Le schéma ''de sécurité'' proposé n'utilise aucune base de confiance entre les nœuds voir aucune primitive cryptographique, et ne traite que le problème d'unicité lié au fonctionnement intrinsèque du protocole prophet. Aucun mécanisme n'est fourni pour contrer les attaques sur le comportement global du réseau, qui ont pour origine l'adressage IP.

II.4 Conclusion sur l'analyse des schémas de sécurité précédents

Par cette analyse, nous démontrons qu'aucune solution suggérée jusqu'ici n'a vraiment résolu le problème de sécurité du service d'auto-configuration d'adresses IP dans les réseaux MANETs. Le Tableau 2.2 récapitule les faiblesses des propositions présentées ci-dessus vis-à-vis des menaces de sécurité mentionnées dans la section II.2. Comme résultat évident, la conception de schémas d'auto-configuration sécurisés doit répondre à des exigences impératives, qui sont détaillées dans la section suivante.

	Buiati et al.	Cavalli et Orset	Wang et al.	Hu et Mitchell	Langer et Kühnert
IP Spoofing Attack	vulnérable	vulnérable	vulnérable	vulnérable	vulnérable
Exhaustion Address Space Attack	vulnérable	vulnérable	-	-	vulnérable
Conflict Address Attack	vulnérable	vulnérable	vulnérable	-	vulnérable
False Conflict Address Attack	-	-	vulnérable	vulnérable	Protected
Sybil Attack	vulnérable	vulnérable	vulnérable	vulnérable	vulnérable
Traffic overload DoS Attack	vulnérable	vulnérable	vulnérable	vulnérable	vulnérable
Pre-authentification	oui, un seul sens	Mutuelle	oui	non	non
Adresse IP liée	non	non	Clé publique hachée	non	non
Détection d'intrusion et accusation	oui	non	non	oui	oui
Traitement du partitionnement/fusion	non	non	non	oui	oui
Autres limitations	Autorité de certification on-line à un saut	Un nombre seuil de voisins à un saut	Schéma Stateless et absence d'une Autorité de Certification pour signer les clés générées	Schéma Stateless et plusieurs hypothèses non réalistes	Schéma Stateless, pas de signature de message et le problème de Diffie Hellman key exchange

Tableau 2.2 – Récapitulatif des faiblesses et limitations des propositions existantes

II.5 Exigences de sécurité pour le service d'auto-configuration

Dans les réseaux MANETs, le niveau de sécurité requis dépendra du contexte du déploiement. Les mécanismes de sécurité qui seront mis en application dans les réseaux militaires, par exemple, sont différents de ceux du déploiement civil ou commercial. Les ressources des nœuds mobiles et le modèle d'adversaire ne sont pas évidemment les mêmes.

Nous considérons ici un modèle d'adversaire fort (actif et dynamique). Les exigences impératives pour concevoir un schéma d'auto-configuration sécurisé et robuste sont comme suit :

E1 :

Les nœuds peuvent joindre ou quitter le réseau MANET dynamiquement ; la seule condition est qu'un nœud doit détenir un certificat valide. Ceci suppose l'existence d'une autorité de certification hors ligne (off-line) pour signer un « certificat de clé publique off-line » pour n'importe quel nœud légitime qui participera au MANET. Pour joindre le réseau, chaque nœud doit détenir son « certificat de clé

publique off-line » et la clé publique de cette autorité off-line pour être capable de vérifier la validité de n'importe quel « certificat de clé publique off-line ». Si un nœud joint le réseau pour la première fois il doit employer son « certificat de clé publique off-line » pour la pré-authentification.

E2 :

Avant toute auto-configuration, une pré-authentification mutuelle doit avoir lieu entre chaque demandeur d'adresse IP et les serveurs de configuration automatique. Le mécanisme de la pré-authentification mutuelle permet aux serveurs d'authentifier le demandeur, ainsi uniquement des nœuds légitimes peuvent participer au réseau, mais également au demandeur d'authentifier les serveurs pour empêcher l'attaque « Man-In-the Middle ».

E3 :

Chaque nœud participant dans le réseau MANET doit pouvoir vérifier à tout moment si une clé publique est révoquée, par conséquent un schéma de révocation est nécessaire dans le réseau MANET. Les nœuds doivent pouvoir révoquer leurs propres clés publiques ou révoquer les clés publiques de nœuds malicieux/compromis, ceci peut être réalisé grâce à un schéma d'accusation [52].

E4 :

L'implémentation de mécanismes pour sécuriser le service d'auto-configuration d'adresses ne devrait pas détériorer la disponibilité du service d'auto-configuration. À n'importe quel instant donné pendant la vie du réseau, un nouveau nœud joignant le réseau MANET devra obtenir avec un protocole sécurisé une adresse IP unique.

E5 :

Pour éviter tout maillon faible (tierce partie ou premier nœud dans le réseau), le service d'auto-configuration doit être initialisé par une coalition de nœuds au lieu d'un seul nœud.

E6 :

Afin de contrôler l'espace d'adressage, le schéma d'auto-configuration doit être stateful. Les adresses IP devraient être assignées exclusivement par le réseau, empêchant des nœuds malveillants à participer librement dans le réseau.

E7 :

L'adresse IP attribuée doit être signée par une autorité certification en ligne (on-line). Ceci permet à un nœud de prouver que son adresse IP a été attribuée par le réseau.

E8 :

L'adresse IP doit être liée à l'identité du nœud pour parer les attaques d'usurpation d'adresse IP et sybile. Ceci semble être la seule solution raisonnable pour empêcher ces attaques. Un certificat est utilisé généralement à cette fin. Selon l'exigence (7), ce certificat doit être signé par une autorité de certification on-line.

E9 :

Pour tous les paquets échangés, les nœuds impliqués doivent vérifier si l'adresse IP de l'émetteur correspond bien à celle qui apparaît dans son certificat on-line, et si également ce certificat est valide et non révoqué.

E10 :

Le schéma doit prendre en considération les problèmes de collisions d'adresses et de sécurité qui peuvent surgir en raison d'un changement de topologie (partitionnement et fusion de réseaux).

II.6 Conclusion

Les réseaux MANETs sont caractérisés par des environnements sans fil ouverts et hostiles, le manque d'infrastructure, des topologies dynamiques et des ressources limitées. En concevant de tels réseaux, plusieurs problèmes intéressants et difficiles surgissent en raison de ces caractéristiques. Ceci a introduit de nouveaux défis comprenant la configuration automatique d'adresse IP, le routage, la sécurité et la QoS, qui ont stimulé l'intérêt considérable des recherches ces dernières années. Dans le cadre de la sécurité, il reste certains problèmes non résolus tels que les problèmes de sécurité relatifs à l'auto-initialisation du réseau et l'auto-configuration. Dans ce chapitre, nous nous sommes concentrés sur ces points. Nous avons présenté un aperçu et une analyse de tous les travaux liés à ce problème, démontrant leurs limitations et faiblesses, ce qui nous a menés à définir les exigences de sécurité impératives surmontant les défauts des conceptions précédentes. Notre but, comme il a été souligné au début de cette thèse, est de développer un protocole sécurisé et robuste, qui accomplira la tâche d'auto-initialisation et de configuration automatique tout en contrecarrant toutes les attaques possibles citées dans ce chapitre, sous hypothèse d'un modèle d'adversaire plutôt fort. Une solution possible à cette issue sera présentée dans le chapitre 4, exploitant les outils récemment publiés de la cryptographie à seuil basée sur le problème du logarithme discret. De toute évidence, la solution proposée a besoin d'un modèle de confiance, c'est la raison pour laquelle une introduction aux modèles de confiance basés sur la cryptographie à seuil adaptés au contexte MANET est présentée au chapitre suivant.

CHAPITRE 3

Modèles de confiance de PKI distribuée et Cryptographie à seuil

III.1 Introduction

Pour assurer la sécurité des réseaux MANETs, en premier lieu et avant tout des schémas d'authentification sont exigés. L'authentification est une procédure essentielle qui permet de vérifier l'authenticité des nœuds participants et leur acceptabilité dans le réseau MANET. Cette authentification n'est possible que sur la base d'un modèle de confiance préétabli reposant sur les systèmes de credentials (password, certificat, carte à puce, ..). Ensuite suivant les besoins, un crypto-système ou un schéma de signature est nécessaire. Dans le chapitre précédent, nous avons défini certaines exigences pour les besoins d'une solution d'auto-configuration sécurisée et robuste en présence d'un adversaire fort. Dans ce chapitre, nous parcourrons un certain nombre de propositions existantes de modèles de confiance répartis pour les réseaux MANETs, à la recherche d'un schéma d'authentification forte qui soit robuste et approprié pour la solution que nous voulons développer.

Les modèles de confiance tiennent une place très importante dans la littérature relative à la sécurité des réseaux MANETs. Il existe deux grands courants pour résoudre le problème de base de confiance et d'authentification dans de tels réseaux. Ces deux groupes de techniques cryptographiques sont définis en adoptant la classification de base utilisée dans la cryptologie, qui distingue les approches symétriques (ou : à clé secrète partagée) et les approches asymétriques (ou à clé publique).

La première se base sur l'établissement d'une clé secrète permettant par la suite l'authentification des participants, la seconde se base sur la cryptographie à clé publique et cherche à s'affranchir du besoin d'une autorité de certification centralisée.

Cependant, pour les besoins de notre solution le modèle de confiance recherché doit permettre également d'établir un schéma de signature (exigence E6, chapitre 2). La notion d'une signature digitale peut s'avérer être l'une des inventions les plus fondamentales et les plus utiles de la cryptographie moderne [53,60,62]. Un schéma de signature fournit un moyen à tout nœud de signer des messages de sorte que les signatures puissent être vérifiées plus tard par n'importe quel autre nœud, et le signataire ne peut pas plus tard nier avoir signé le message (non répudiation). Dans la cryptographie symétrique, la clé étant connue de deux ou plusieurs nœuds, cette technologie n'offre pas de support direct pour la signature. Ainsi, seule la cryptographie à clé publique peut fournir de tels services. La synthèse qui suit portera uniquement sur les approches asymétriques.

III.2 Modèles de Confiance basés sur la cryptographie asymétriques

Les systèmes cryptographiques asymétriques sont les moyens les plus couramment utilisés dans la pratique pour assurer les services d'authentification, d'intégrité et de non répudiation [62]. Dans cette section, nous décrivons différents modèles de confiance pour les réseaux MANETs qui sont basés sur la cryptographie asymétrique. Les clés publiques sont employées pour l'authentification de nœuds et pour l'établissement de clés de session. Les clés de session sont alors employées dans un schéma de chiffrement symétrique pour fournir la confidentialité des communications entre les nœuds authentifiés (canaux sécurisés). La cryptographie à clés publiques nécessite l'emploi d'une architecture centralisée telle qu'une PKI (public key infrastructure), reposant principalement sur une autorité de certification (CA) dont les principales tâches sont la création et le renouvellement de certificats, la publication des clés publiques, et la révocation des certificats sur date de péremption, perte, ou compromission. Dans le contexte des réseaux MANETs qui sont par nature sans infrastructure, l'absence d'une autorité de certification centralisée est le problème principal dans la mise en application des protocoles asymétriques.

Nous distinguons trois catégories de modèles de confiance basés sur la cryptographie asymétrique:

- avec autorité de certification et avec utilisation de certificats : modèle de PKI basée sur CA distribuée,
- avec autorité de certification et sans utilisation de certificats : modèle de certificat basé sur l'identité et modèle de clé publique auto-certifiée,
- sans autorité de certification mais avec utilisation de certificats : modèle de PKI auto-organisée.

III.2.1 Modèle de PKI basée sur CA distribuée

La PKI a été identifié en tant qu'une des méthodes les plus réussies d'authentification pour les réseaux dynamiques et pour fournir la base de confiance [54-59,61]. Cependant, fournissant une telle fonctionnalité dans les réseaux MANETs n'est pas une tâche facile principalement en raison de l'absence de toute infrastructure. La première problématique soulevée était donc l'adaptabilité de l'architecture PKI au contexte ad hoc.

Plusieurs travaux ont été proposés pour émuler un CA on-line en distribuant son rôle entre les nœuds participants au réseau [67,68]. L'idée est basée sur le fait qu'un CA ne devrait pas être représenté par un seul nœud, devenant le point vulnérable du réseau; de protection physique faible, il pourrait être compromis relativement facilement par un adversaire. Celui-ci peut alors signer n'importe quel certificat incorrect en utilisant la clé du CA pour usurper l'identité de n'importe quel nœud ou pour révoquer n'importe quel certificat valide. Si le nœud faisant office de CA est inaccessible, le service de certification devient indisponible. Une approche standard pour améliorer la disponibilité d'un service est la reproduction. Mais une reproduction naïve du CA rend le service plus vulnérable, la compromission de n'importe quel nœud du réseau, qui possède la clé privée du CA, pourrait mener à l'effondrement du système entier.

Nous distinguons dans ce modèle deux solutions. La première est basée sur une CA partiellement distribuée (Partially Distributed Certificate Authority) [54-58], où certains nœuds sont choisis pour émuler le rôle du CA. La seconde est basée sur une CA distribuée à travers tout le réseau (Fully Distributed Certificate Authority) [59,61], où tous les nœuds peuvent participer dans l'émulation du rôle du CA. Les deux solutions sont basées sur le partage de secret et la cryptographie à seuil.

a) Modèle de CA partiellement distribuée

Dans le premier modèle de CA partiellement distribuée présenté initialement par Zhou et Haas dans [54] et développé ensuite dans le système COCA [55], le réseau contient ***n*** nœuds appelés serveurs et plusieurs nœuds appelés combineurs. La clé publique est connue à tous les nœuds tandis que la clé privée est divisée en ***n*** parts en utilisant un schéma de cryptographie à seuil (***k***, ***n***). Ces parts sont assignées aux ***n*** serveurs. Pour signer un certificat dans un tel schéma, chaque serveur emploiera sa part pour produire une signature partielle et puis l'expédier à un nœud combineur qui se charge de combiner les signatures partielles afin de produire une signature régulière identique à celle qu'aurait générer une CA centralisée.

L'application de la cryptographie à seuil assure que le système peut tolérer un certain nombre ***k-1*** < **n** de serveurs compromis dans le sens qu'au moins ***k*** signatures partielles sont nécessaires pour reconstituer une signature régulière correcte. Cependant, cette proposition suppose qu'il y a une

autorité qui désigne et initialise au démarrage du réseau les serveurs et la partie des nœuds qui doivent se comporter comme combineurs. Elle ne décrit pas également comment la valeur ***k*** est choisie, comment un nœud peut entrer en contact d'une façon sécurisée et efficace avec ***k*** serveurs quand les serveurs sont dispersés à travers le réseau entier, et comment les rôles de serveur et de combineur sont distribués et employés, ou ce que serait le cas si le nœud ne pourrait pas trouver assez de serveurs ou un combineur pour produire la signature. Une autre faiblesse de la solution est le manque d'un mécanisme de révocation de certificat. N'importe quelle solution basée sur des certificats devrait, vu le risque de compromission dans les réseaux MANETs, fournir un tel mécanisme.

Seung Yi et Robin Kravets [56] ont proposé ensuite de distribuer la confiance aux nœuds ayant une forte sécurité physique et une bonne puissance de calcul, particulièrement dans un environnement hétérogène composé de nœuds ayant différentes caractéristiques. Ces nœuds sont appelés MOCAs (Mobile Certificate Authorities). Le schéma MOCA déplace également la fonction de combineur [54] des serveurs de CA aux nœuds clients du service. Ceci améliore la disponibilité du système puisque les nœuds ne dépendent plus des serveurs de CA pour combiner les signatures partielles de certificat. Un protocole de certification MOCA (MOCA certification Protocol, MP) est proposé par [56] pour fournir des communications décisives et efficaces entre les nœuds clients et les serveurs MOCAs. Le protocole MP unicast la demande de certificat à (***k*** + α) serveurs MOCAs spécifiques pour augmenter la probabilité de recevoir au moins ***k*** réponses. La recherche des serveurs MOCAs est semblable à celle utilisée par les protocoles de routage réactifs tels qu'AODV et DSR. Le protocole MP maintient ses propres tables de routage. Les entrées de table de routage MOCA sont rafraichies pour une réutilisation potentielle. Le protocole MP emploie l'inondation quand il n'y a pas assez de routes aux serveurs MOCAs dans le cache.

Le protocole MP ne précise pas comment les nœuds peuvent découvrir d'une manière sécurisée les routes aux serveurs MOCAs, ni qui juge le niveau de sécurité, la façon dont les serveurs MOCAs sont choisis, et la façon de s'assurer qu'ils sont distribués uniformément. Le protocole MP maintient ses propres tables de routage parallèlement à un protocole de routage dédié, ce qui représente un gaspillage de la bande passante.
En outre, la mise en cache allège le problème d'overhead de communication dans une certaine mesure quand le réseau reste statique, de sorte que les routes cachés soient valables une période relativement longue. Cependant, dans une topologie dynamique cette optimisation sera insuffisante. Similairement à [55], cette solution souffre du manque d'un mécanisme de révocation.

b) Modèle de CA totalement distribuée

Jiejun Kong et al. ont proposé une approche de CA totalement distribuée utilisant la cryptographie à seuil [59]. Par opposition aux schémas de CA partiellement distribués, ce modèle fournit une distribution totale permettant à tous les nœuds d'obtenir une part de la clé privée du CA. Une coalition de k voisins d'un saut forme la fonctionnalité locale de CA. Elle n'a besoin d'aucun protocole de routage, mais uniquement d'une densité de k ou plus de nœud voisins d'un saut. La mobilité peut aider à localiser le nombre requis de nœuds de CA. Les nœuds sont dignes de confiance dans le réseau entier quand ils reçoivent un certificat valide. N'importe quel nœud tenant un certificat valide peut obtenir une part de la clé privée du CA. Les auteurs ont proposé un protocole original d'auto-initialisation (shuffling) pour manipuler l'adhésion dynamique de nœud et les mises à jour des parts du secret. Chaque nœud peut être initialisé par ses voisins. La nouvelle part secrète est calculée en ajoutant les parts partielles reçues d'une coalition de k voisins. Une fois initialisé, un nœud est qualifié pour être un membre de coalition pour servir son voisinage.

À l'état initial du système, un ensemble de nœuds recevront leurs certificats d'un distributeur (dealer). Une fois que $l \geq k$ nœuds ont été initialisés, le dealer est enlevé. Dans ce schéma, on suppose que le service de certification est fourni dans des voisinages d'un saut. En limitant les demandes des services du CA à un saut on économise la bande passante et on facilite le passage à l'échelle.

Le schéma est donc entièrement localisé et disponible partout, il fournit un service omniprésent pour les nœuds par distribution de la fonctionnalité du CA à chaque voisinage local. Une coalition de voisins peut ''à tout moment'' servir de CA et conjointement fournir des services de certification pour un nœud client.

Cependant, similairement à la proposition [54], les premiers nœuds doivent être initialisés par une autorité de confiance. En plus de cet inconvénient, le niveau de sécurité du système dépend du paramètre k. Puisque chaque nœud fait office de serveur de CA, et la compromission de n'importe quel nombre k d'entre eux révélera la clé de signature privée, ceci met en danger réellement la sécurité pour un petit k relativement au nombre total de nœuds. Quand k est assez grand, cela résulte en une sécurité accrue mais une tolérance aux fautes réduite.
Les auteurs supposent que le service de certification est fourni dans des voisinages d'un saut, or il n'est pas toujours possible de trouver k nœuds dans un tel voisinage, la disponibilité n'est donc pas toujours garantie.
Le schéma proposé souffre de l'overhead de communication, en effet le service de certification exige l'interaction de k nœuds, ce qui remet en cause son efficacité. Pour obtenir un certificat, un nœud doit s'identifier à k nœuds. Les auteurs proposent que le processus d'identification soit effectué par contact

direct. Ceci semble être une hypothèse très restrictive puisqu'il exige des ***k*** nœuds d'être dans le voisinage direct d'un nœud client. Toutefois, nous pouvons alléger cette hypothèse en exigeant simplement un canal sécurisé.

III.2.2 Modèle de certificat basé sur l'identité

La cryptographie basée sur l'identité (Identity Based Cryptography, IBC) a été introduite par Shamir [63] comme alternative au paradigme traditionnel d'infrastructure à clé publique PKI. La motivation était de se passer du besoin des certificats numériques, signés par une autorité de certification, qui lient l'identité d'un utilisateur avec sa clé publique. En effet, pour éviter les attaques d'usurpation, la validité du certificat numérique correspondant doit être vérifiée avant l'utilisation de la clé publique pour chiffrer ou pour vérifier une signature. L'infrastructure requise pour contrôler de tels certificats est très coûteuse et ceci est particulièrement problématique, comme nous venons de le voir, dans le cas des réseaux MANETs. Dans la cryptographie basée sur l'identité, la clé publique de chaque utilisateur peut être dérivée, d'une manière publique et efficace, directement de son identité. Par conséquent, le lien entre l'identité et la clé publique est implicitement établi.

Puisque les clés publiques sont prédéterminées dans les schémas IBC, les clés privées sont dérivés des clés publiques correspondantes. Pour cette raison, les schémas IBC exigent une certaine entité master pour produire et distribuer des clés privées pendant l'initialisation de tous les nœuds du réseau. L'utilisateur doit entrer en contact avec cette entité master pour obtenir sa clé privée. L'entité master a une paire de clés privée/publique appelés clés master, la clé privée master est utilisée pour calculer les clés privées des utilisateurs. L'inconvénient principal de ce schéma est que l'entité master connait les clés privées de tous les utilisateurs. L'entité master délivre les clés privées aux utilisateurs à travers un canal sécurisé. Pour limiter la période de validité d'une clé publique IBC, une date de validité peut être facilement incluse dans la clé publique elle-même, par exemple par concaténation.

Généralement, le rôle spécial de l'entité master dans les schémas IBC est considéré comme un inconvénient. Dans le contexte MANET, quelques schémas IBC [18, 64, 65] ont été proposés, basés sur une entité master distribuée en utilisant la cryptographie à seuil. Ces schémas se basent en général sur les fonctions bilinéaires (bilinear pairing) [66]. Un nœud devra obtenir sa clé privée en combinant ***k*** parts calculées et fournies par ***k*** parmi les ***n*** nœuds originaux détenant collectivement la clé privée master.

L'inconvénient majeur des schémas IBC est la condition d'un canal sécurisé (confidentialité et authenticité) entre les nœuds serveurs de génération de clés et chaque nœud de réseau pour la distribution des clés privées.

Il convient de noter que cette approche n'authentifie pas les identités des nœuds, ceci est problématique dans le sens où un nœud malveillant peut usurper l'identité d'un autre nœud et entrer en contact avec ***k*** nœuds serveurs pour obtenir la clé privée correspondante. De cette façon ce nœud malveillant pourra chiffrer et déchiffrer des messages au nom du nœud usurpé. En conclusion, fournissant la révocation dans des schémas IBC est considéré comme une issue.

III.2.3 Modèle de clé publique auto-certifiée

Dans les schémas à clé publique auto-certifiée [19], les certificats sont inclus dans les clés publiques elles-mêmes. Contrairement aux schémas IBC où l'identité d'un nœud est directement employée comme clé publique, dans le modèle de clé publique auto-certifiée la clé publique d'un nœud est construite à partir de son identité. Par conséquent, les clés publiques doivent être échangées avant la communication.

Dans ce modèle, chaque nœud possède préalablement une paire de clés privée et publique. Une autorité de certification CA est exigée pour publier les clés publiques auto-certifiées. Le CA produit les clés publiques auto-certifiées en utilisant comme entrées la clé publique du nœud, son identité, et la clé privée du CA. Noter que le CA ne connait pas les clés privées des nœuds. Les nœuds devront ensuite utiliser leurs clés publiques auto-certifiées pour toutes les authentifications dans le réseau MANET. Les mécanismes de chiffrement et de signature dans ces schémas sont différents de ceux des schémas à clé publique classique, du fait que les clés publiques auto-certifiées et les clés privées associées ne correspondent pas directement les uns avec les autres. L'authenticité des clés publiques est fournie par les clés elles-mêmes. Il n'y a aucun besoin de certificats ou de tout autre mécanisme pour fournir un canal authentique. Cette approche aide à économiser un certain espace de bande passante et de mémoire, parce que des certificats n'ont pas besoin d'être transmis et stockés. En dépit de ces avantages, aucune solution à clé publique auto-certifiée appropriée aux réseaux MANETs n'a encore été proposée.

III.2.4 Modèle de PKI auto-organisée

Dans ce modèle proposé par Hubaux et al. [20] on ne se sert absolument d'aucune infrastructure, ni même pendant la configuration initiale. Comme dans PGP [60,69], les clés publiques et les clés privées sont créés par les nœuds eux-mêmes. Chaque nœud agit en tant que CA. Les nœuds délivrent des certificats entre eux basés sur la connaissance personnelle. Chaque nœud possède une paire de clés privée et publique et établit des certificats pour les nœuds en qui il a confiance. Lorsque deux nœuds veulent communiquer sans connaissance au préalable l'un de l'autre, ils s'échangent leurs listes de certificats et vont essayer de créer une chaine de confiance entre eux.

Les certificats sont délivrés seulement pendant une période spécifique, la période de validité est inscrite dans le certificat. Avant qu'elle expire, le certificat est mis à jour par le nœud qui avait délivré le certificat.
Des mécanismes de construction de base de données locale sont employés pour contenir les certificats de nœuds, de sorte que n'importe quelle paire de nœuds dans le réseau puisse établir, avec une probabilité élevée, une chaîne de confiance entre eux. Au commencement, chaque nœud a une base de données locale contenant les certificats délivrés par le nœud lui-même et les certificats fournis pour lui par d'autres nœuds. Périodiquement, des certificats des voisins sont demandés et la base de données est mise à jour en ajoutant de nouveaux certificats.

Le modèle apparenté de confiance entre les nœuds est représenté par un graph orienté G (V, E) appelé graphe de certificats. V et E représentent respectivement l'ensemble des sommets (nœuds) et l'ensemble des arrêtes orientées (certificats) du graphe. Ainsi, l'existence d'une arrête entre deux sommets ***u*** et ***v*** dans le graphe de certificats signifie que le nœud ***u*** a produit un certificat pour le nœud ***v***. L'existence d'une chaîne de confiance entre deux des nœuds du réseau MANET est ainsi représentée par un itinéraire direct entre les deux sommets du graphe, représentant les deux nœuds concernés.

Cette technique distribuée d'authentification a des garanties probabilistes, étant donné que l'existence d'une chaîne de confiance entre deux nœuds dans le graphe n'est pas assurée. En outre, le stockage distribué des certificats de nœud produit des overhead élevés, qui rendent la vraie applicabilité de cette approche difficile à grande échelle. En plus, des membres malveillants peuvent produire des certificats incorrects et les intégrer dans le graphe de certificats. Aucun algorithme pour la révocation n'est décrit dans [68].
D’autre part, une certaine période est nécessaire pour peupler le graphe de certificats, qui dépend complètement du comportement et de la mobilité des nœuds, et il n'y a aucune garantie que le graphe de certificats résultant sera assez dense pour authentifier toutes les clés publiques dans le réseau. En conclusion, la validité d'une chaîne de certificat dépend seulement de la fidélité des nœuds dans cette chaîne, ceci n’est pas approprié aux applications où des niveaux importants de responsabilité et de sécurité sont exigés.

III.2.5 Discussion

Parmi les exigences pour concevoir un schéma d’auto-configuration sécurisé et robuste sous l’hypothèse d’un modèle d’adversaire fort figure l’authentification mutuelle forte des participants et l’authenticité des messages. Cela impose un choix d’un système cryptographique asymétrique avec authentification par certificat. D’autres parts, un schéma de révocation est exigé pour isoler les nœuds

malveillants et révoquer leurs certificats. Parmi les solutions asymétriques existantes conçues pour les réseaux MANETs seul le modèle de PKI répond à cette exigence. Nous recherchons, en outre, une solution garantissant la disponibilité, le modèle totalement distribué répond parfaitement à notre besoin.

Dans la suite de ce travail, nous nous concentrons sur le modèle de confiance de PKI basée sur une CA « on line » totalement distribuée, utilisant le principe de la cryptographie à seuil. Comme nous venons de l'expliquer, ce modèle répond parfaitement aux exigences soulignées dans le chapitre précédent. De ce fait, nous avons jugé utile de présenter dans la section suivante un survol de la cryptographie à seuil : ses avantages, ses problèmes et ses outils.

III.3 Généralités sur la Cryptographie à seuil

La cryptographie à seuil a suscité une attention considérable en littérature [70-80]. Dans un système cryptographique à clé publique à seuil (***k, n)*** appliqué à un réseau MANET où ***n*** est la taille du réseau et ***k*** désigne le seuil, la clé privée du réseau (le secret) est partagée en ***n*** parts, à chaque nœud est attribuée une part de la clé privée. Le service de l'autorité de certification est réalisé d'une manière transparente par tout sous-ensemble comportant un nombre de nœuds au moins égal au seuil ***k*** (***k*** parmi ***n***). Un adversaire doit compromettre donc plus que (***k-1***) nœuds afin d'apprendre le secret, et corrompre au moins ***n***-(***k-1***) nœuds pour mettre en défaut la disponibilité du service.

Ainsi, pour une générer par exemple une signature d'un certificat, une coalition de ***k*** nœuds est désignée pour cette tâche, chaque nœud appartenant à cette coalition génère une signature partielle en utilisant sa part du secret, et transmet le résultat à un combineur qui sera chargé d'assembler les portions de signature pour calculer la signature globale du certificat.

Un premier avantage des schémas cryptographiques à seuil est qu'ils sont utilisés pour des applications critiques où l'on souhaite éviter qu'une seule entité puisse déchiffrer ou signer. Un autre avantage est de résister à des adversaires forts qui peuvent corrompre plusieurs nœuds, disons (***k-1***) parmi ***n***, et avoir accès au contenu de leurs mémoires. Il est réaliste de considérer ces adversaires puisque les intrusions réseaux sont faciles à monter, principalement à cause de l'architecture TCP/IP ouverte. Ainsi, un adversaire peut, suivant les systèmes d'exploitation utilisés, pénétrer sans trop de difficulté dans un nœud et obtenir toutes les clés cryptographiques présentes.

III.3.1 Bases de la théorie des nombres appliquée à la cryptographie asymétrique

- **Groupe modulo**

Soit Z_N le groupe additif modulo N (où N est un entier) :

$$Z_N = \{0, 1, 2, \dots, N\}$$

L'ordre d'un groupe est son cardinal. Ainsi l'ordre de Z_N est N+1.

Le groupe multiplicatif modulo N de Z_N, noté Z_N^* est :

$$Z_N^* = \{i \in Z_N \ / pgcd(i, N) = 1\}$$

- **Fonction d'Euler**

La fonction d'Euler $\Phi(N)$ est égale au nombre d'entiers positifs plus petits que N qui sont relativement premiers à N. L'ordre de Z_N^* est donc égal à $\Phi(N)$.

En particulier, si p est premier alors :

$$Z_p^* = \{1, 2, \dots, p-1\}$$

$$\Phi(p) = p - 1 \quad \textbf{(III.1)}$$

Si $N = pq$, p et q sont premiers, alors :

$$\Phi(N) = \Phi(p)\Phi(q) = (p-1)(q-1) \quad \textbf{(III.2)}$$

Dans une exponentiation modulaire (modulo N), les exposants doivent être pris modulo $\Phi(N)$, soit :

pour tout $a, x \in Z_N^*$, on a : $a^{x(mod\Phi(N))} = a^x (modN)$

- **Théorème d'Euler**

Pour tout $a \in Z_N^*$, on a :

$$a^{\Phi(N)} = 1 \ (modN) \quad \textbf{(III.3)}$$

- **Petit Théorème de Fermat**

Il s'agit d'un cas particulier du théorème d'Euler, pour tout $a \in Z_p^*$, où p est premier on a :

$$a^{p-1} = 1 \ (modp) \quad \textbf{(III.4)}$$

- **Générateur et groupe cyclique**

Soit $a \in Z_N^*$, l'ordre de a est le plus petit entier positif m pour lequel:

$$a^m = 1 \ (modN) \quad \textbf{(III.5)}$$

Soit $g \in Z_N^*$, si l'ordre de g est $\Phi(N)$, alors g est un générateur de Z_N^* .

Lorsqu'un groupe Z_N^* a un générateur, on dit qu'il est cyclique.

Cas particulier : si p est premier alors Z_p^* est cyclique ; soit g est un générateur de Z_p^* alors :

Pout tout élément $y \in Z_p^*$, il existe un élément unique x $(0 \leq x \leq p-2)$ tel que : $y = g^x (modp)$

Cet unique x est dénoté $log_g(y)$ et appelé le logarithme discret de x dans la base g.

- **Fonction à sens unique**

Soient p un nombre premier et g est un générateur de Z_p^*, et soit la fonction f définie par :

$$f : Z_p \to Z_p^*$$
$$x \to y = g^x (mod p)$$

La fonction f est conjecturée pour être à sens unique pour certaines conditions sur p, c'est-à-dire qu'il n'existe aucun algorithme efficace pour l'inverser (calculer le logarithme discret de x dans la base g) pour des valeurs assez grandes des paramètres.

Une propriété intéressante de cette fonction : f est un homomorphisme de groupe

Soit pour tout $a, b \in Z_p$ on a : $g^{a+b} = g^a . g^b (mod p)$.

III.3.2 Théorie de la complexité

La cryptologie fait appel à la théorie de la complexité [81,82] pour évaluer la sécurité des schémas cryptographiques, et savoir quelles ressources sont nécessaires pour en mettre en défaut leur sécurité. La théorie de la complexité permet de rendre compte des ressources nécessaires à la résolution d'un problème donné et de classer les problèmes suivant leur difficulté. Les ressources généralement prises en compte sont le temps de calcul, et parfois l'espace mémoire.

Etant donné un algorithme résolvant un problème défini à partir d'une entrée de taille t. La complexité de l'algorithme représente le nombre d'opérations nécessaires pour résoudre le problème en fonction de t. Généralement les complexités sont asymptotiques (t $\to\infty$).

On dit qu'un algorithme est polynomial si sa complexité dans le pire cas est inférieure à un polynôme en la taille des entrées. Les algorithmes polynomiaux s'identifient généralement aux algorithmes efficaces dans la pratique, permettant d'obtenir rapidement un résultat.

III.3.2.1 Classes importantes de complexité

En général, les algorithmes sont classés dans la théorie de complexité en trois catégories : polynomiale, sous-exponentielle et exponentielle. La fonction $L_t[c, \alpha] = \exp(ct^{\alpha}(logt)^{1-\alpha})$ pour $\alpha \in [0,1]$, par exemple, est :

- polynomiale si $\alpha = 0$,
- exponentielle si $\alpha = 1$,
- sous-exponentielle et super-polynomiale sinon.

III.3.2.2 Classes de problèmes décisionnels

Un domaine important de la théorie de la complexité concerne l'évaluation des problèmes de décision, soit l'ensemble des problèmes qui ont pour réponse Oui ou Non. Beaucoup de problèmes dont l'énoncé n'est pas décisionnel peuvent se ramener à des problèmes décisionnels.

La complexité des algorithmes a abouti à une classification de ces problèmes en fonction des performances des meilleurs algorithmes connus qui les résolvent. Il faut noter que cette classification est indépendante des caractéristiques techniques des ordinateurs.

a) Classe P

La classe P est l'ensemble des problèmes de décision qui sont solvables par un algorithme à temps polynomial. Autrement dit, un problème de décision est dans P s'il peut être décidé par un algorithme déterministe en un temps polynomial par rapport à la taille de l'instance.

b) Classe NP

NP signifie “Nondeterministic Polynomial-time”. La classe NP réunit les problèmes de décision pour lesquels la réponse *oui* peut être décidée par un algorithme non-déterministe en un temps polynomial par rapport à la taille de l'instance. Nous avons la conjoncture $P \neq NP$.

c) Problème NP-Complet

Un problème décisionnel L est dit NP-complet s'il appartient à la classe NP et si tout problème de la classe NP se réduit en temps polynomial à L. Les problèmes NP-complets sont les problèmes les plus difficiles de la classe NP au sens où ils sont au moins aussi difficiles que tout autre problème dans NP.

III.3.3 Modèle de sécurité des schémas cryptographiques à seuil

Un modèle de sécurité permet de modéliser la vie réelle d’un système et les interactions des différents acteurs. On classe les schémas cryptographiques à seuil en fonction de différents critères comme les propriétés de sécurité que tente d’attaquer un adversaire, les hypothèses faites sur les canaux de communication entre les nœuds ou encore la puissance des adversaires.

III.3.3.1 Propriétés de sécurité

Les deux propriétés que tente de mettre en défaut un adversaire contre un système cryptographique à seuil sont : la sécurité et la robustesse.

a) Sécurité

Il y a deux approches fondamentales dans l’étude de la sécurité d’un système cryptographique : la sécurité inconditionnelle ou sécurité au sens de la théorie de l’information de Shannon et la sécurité calculatoire.

- Sécurité inconditionnelle :

  Ceci mesure la sécurité du système sans borne sur la quantité de calcul que l'attaquant est capable de faire. Un système cryptographique est inconditionnellement sûr s'il ne peut être cassé, même avec une puissance de calcul infinie. Shannon a prouvé qu'une telle sécurité nécessitait que la longueur de la clé soit égale au moins celle du texte chiffré.

- Sécurité calculatoire :

  La sécurité calculatoire mesure la quantité de calcul nécessaire pour casser un système. On dira qu'un système cryptographique est sûr au sens de la théorie de la complexité si le meilleur algorithme pour le casser est au mieux super-polynomial en temps.

La cryptographie moderne abandonne la prétention que l'adversaire dispose de ressources informatiques infinies, et suppose à la place que l'adversaire est est polynomialement borné.

Hypothèses Cryptographiques

Dans la conception de systèmes cryptographiques pratiques, on cherche des schémas qui soient prouvés sûrs, ceci consiste à prouver la sécurité en montrant qu'un schéma ne peut pas être cassé sans l'aide d'une quantité importante de ressources de calcul. Souvent, on fait des hypothèses basée sur des problèmes difficiles au sens de la théorie de la complexité, et on utilise l'état de l'art des meilleurs algorithmes connus qui résolvent ces problèmes difficiles pour estimer les paramètres de sécurité des schémas développés.

La cryptologie à clé publique a su tirer parti de ces problèmes difficiles. On cherche à y mettre en évidence des algorithmes cryptographiques ayant, d'un côté une complexité petite, fonction polynomiale de la taille de la clé, et d'un autre côté une sécurité essentiellement équivalente à la résolution d'un problème pour lequel aucun algorithme de complexité polynomiale en la taille de la clé n'est connu.

Parmi les problèmes cryptographiques réputés difficiles, nous citons : le problème de factorisation et le problème du logarithme discret. Ces problèmes sont dans la classe NP.

- Problème de factorisation :

  Le problème de factorisation d'un nombre entier (Factorization Problem, FP) est le suivant : étant donné un nombre composé N qui est le produit de deux grands nombres premiers p et q, trouver p et q. Tandis que le calcul du produit de grands nombres premiers est une tâche relativement facile, le problème de factoriser ce produit (si p et q sont soigneusement choisis) est considéré difficile au sens de la théorie de la complexité. Basé sur la difficulté de ce problème, Rivest, Shamir et Adleman [83] ont développé le système cryptographique à clé publique RSA. Un autre système

cryptographique à clé publique dont la sécurité repose sur l'intractabilité du problème FP est dû à Rabin et à Williams [84, 85], (Racine carrée dans Z_N si N est composite**)**. Les meilleurs algorithmes connus pour la factorisation d'entiers N étant de complexité asymptotique sous-exponentielle.

- Problème du logarithme discret :

Le problème du logarithme discret (Discrete Logarithm Problem, DLP) est le suivant : Etant donné un nombre premier p très grand, un générateur $g \in Z_p^*$, et $y \in Z_p^*$, trouver $x \in \{0, 1, .., p-2\}$ tel que $y = g^x(modp)$. Il est conjecturé qu'il existe des groupes tels que le groupe multiplicatif et le groupe des courbes elliptiques, dans lesquels la solution du problème DLP est difficile au sens de la théorie de la complexité.

Basé sur la difficulté de ce problème, Diffie et Hellman [86] ont proposé en 1976 le schéma « DH key agreement » bien connu et qui porte leurs noms. Le problème de Diffie-Hellman (Diffie-Hellman Problem, DHP) est de calculer $g^{x_1 x_2}(modp)$ connaissant $g^{x_1}(modp)$ et $g^{x_2}(modp)$.

Depuis lors, on a proposé d'autres systèmes cryptographiques dont la sécurité dépend du problème DLP, incluant : le crypto-système et le schéma de signature d'ElGamal [87], le schéma de signature DSA [88], celui de Schnorr [89], et celui de Nyberg-Rueppel [90]. Il faut noter que les algorithmes les plus efficaces connus à ce jour ont une complexité asymptotique semblable au problème FP, c'est-à-dire sous-exponentielle.

b) Robustesse

Un système cryptographique à seuil est exécuté par ***n*** nœuds. Ils peuvent être honnêtes ou corrompus. Un nœud corrompu peut arrêter son exécution, dévier son exécution de sa spécification (attaque byzantine).

La robustesse permet de garantir que même en présence de nœuds corrompus, le système continue à fonctionner correctement. Comme un adversaire doit compromettre donc plus que (***k-1***) nœuds afin d'apprendre le secret, et corrompre au moins ***n***-(***k-1***) nœuds pour mettre en défaut la disponibilité du service, alors un système cryptographique (***k,n***) robuste doit tolérer au plus ***k-1 < n/2***, soit ***n > 2k-1.***

III.3.3.2 Hypothèses sur le canal de communication

Les modèles de communication diffèrent en fonction de deux critères : l'existence de canaux secrets de communication reliant chaque paire de nœuds et l'existence d'un canal de broadcast qui permet de transmettre le même message à plusieurs destinataires en même temps. Les nœuds connectés à ce canal y compris les nœuds malicieux peuvent lire d'une manière instantanée chaque message diffusé. Suivant les cas, nous ferons l'hypothèse de l'un ou l'autre.

III.3.3.3 Hypothèses sur l'adversaire

En général deux modèles sont définis :

- Adversaire passif : il est capable de lire les informations enregistrées dans les mémoires des nœuds victimes et voir le contenu des messages échangés, et tente uniquement de contredire la sécurité du système.

- Adversaire actif : il s'agit d'un adversaire byzantin qui en plus de la lecture des informations, il est capable de contrôler le comportement du système afin de le dévier de sa spécification (empêcher la terminaison d'une opération cryptographique, altérer ou détruire les messages échangés…), il s'attaque donc à la robustesse du système.

Les adversaires sont aussi classés suivant les ressources de calcul disponibles. Comme il a été déjà précisé, dans le cas d'une sécurité calculatoire, l'adversaire est supposé polynomialement borné. On peut distinguer également des adversaires statiques et adversaires adaptatifs ou mobiles. Contrairement à un adversaire statique qui corrompt les nœuds à l'initialisation du système, un adversaire adaptatif/mobile peut décider de corrompre n'importe quel nœud pendant la durée de vie du réseau. Dans la suite de ce mémoire, nous considérerons un modèle d'adversaire fort celui représenté par un adversaire actif et mobile.

III.4 Problèmes à résoudre pour les schémas cryptographiques à seuil

Certains problèmes inhérents aux systèmes cryptographiques à seuil ont été largement discutés dans la littérature et méritent d'être revus. Nous envisageons d'utiliser les outils développés pour leurs solutions dans notre approche.

III.4.1 Fraude du distributeur : protocoles VSS et PVSS

En général, les schémas proposés à l'origine pour le partage du secret (clé privée) utilisent un distributeur unique appelé distributeur de confiance (trusted dealer) qui génère la clé secrète, la partage et envoie une part de la clé à chaque nœud. Tous les nœuds lui font confiance pour qu'il ne dévoile pas la clé qu'il a générée. En réalité ceci est problématique, mais admettons que le distributeur peut être déconnecté du réseau après l'initialisation des parts secrètes, ceci empêche l'adversaire de le cibler. Cependant, un autre problème se pose, si le distributeur triche et génère un mauvais partage, rien ne garantit que les nœuds puissent reconstruire la clé. Ainsi, la disponibilité du service risque d'être mise en défaut.

La solution à ce problème est l'utilisation d'un schéma de partage de secret vérifiable (Verifiable secret Sharing, VSS). Celui-ci consiste à rajouter des messages permettant aux nœuds de vérifier si le

distributeur de confiance agit correctement dans la phase de partage. Il doit alors prouver que les parts transmises à chaque nœud permettent de reconstruire la clé. Les schémas VSS prennent le nom de partages de secret publiquement vérifiables (PVSS) quand non seulement les nœuds participants peuvent vérifier que le dealer a correctement agi mais qu'un nœud externe quelconque le peut également.

III.4.2 Nécessité de protocoles distribués

Bien qu'il existe des protocoles permettant de partager et reconstruire une clé par une seule entité, la reconstruction et l'utilisation de cette clé dans un seul nœud risque de compromettre sa protection.

Le calcul MPC (Multiparty Computation) [91] ou cryptographie interactive concernant l'étude générale des calculs sécurisés entre plusieurs entités, apporte une solution à ce problème. Elle fournit la construction de schémas cryptographiques distribués, ou partage des fonctions telles que la génération de la clé privée, les crypto-systèmes à seuil (chiffrement et déchiffrement) et la signature à seuil.

III.4.3 Attaques répétées dans le cas d'un MANET à vie longue : Partage de secret proactif

Dans le cas d'un modèle d'adversaire fort, si l'adversaire a suffisamment de temps, il peut attaquer les nœuds un à un, et compromettre la sécurité du réseau. Ce risque est particulièrement problématique dans les réseaux MANETs qui sont déployés pendant de longues périodes de temps.

Une solution à ce problème est la sécurité proactive [80]. Elle se base sur la régénération périodique des parts secrètes de telle sorte que l'information apprise pendant les périodes passées n'aide pas l'attaquant pendant la période suivante. On considère des adversaires qui peuvent corrompre tous les nœuds mais seulement k nœuds par période de temps appelée « fenêtre de vulnérabilité ». Avant que le seuil des nœuds attaqués ne soit atteint, le réseau va repartager la clé, et attribuer de nouveaux parts pour les nœuds. Le partage de secret proactif offre une protection à long terme d'un réseau contre des attaques répétées conduites par des adversaires forts et déterminés.

III.5 Partage de secret à seuil

Le partage de secret constitue la base pour la cryptographie à seuil. L'idée générale du partage de secret est de distribuer un secret parmi un groupe d'entités de sorte que seulement des collections pré-désignées de participants puissent reconstituer le secret en combinant collectivement leurs parts du secret.

Dans un schéma de partage de secret (Secret Sharing Scheme, SSS) [92,93], on définit une structure d'accès sur un ensemble de n participants comme l'ensemble de tous les sous-ensembles (parties de {1,

. . . ***n***}) qui sont conçus pour reconstruire le secret. Les éléments de la structure d'accès sont désignés sous le nom de parties autorisées et le reste sont les parties non autorisées. Une propriété de ces structures est la monotonie : c'est-à-dire, soit **S** est une structure d'accès : si $\mathbf{A} \in \mathbf{S}$ et $\mathbf{A} \subseteq \mathbf{B}$ alors $\mathbf{B} \in \mathbf{S}$. Les schémas de partage de secret qui n'indiquent aucune information sur le secret partagé aux entités non autorisées sont dit parfaits.

Les schémas de partage de secret les plus largement étudiés sont les schémas à seuil (***k, n***). Dans ces schémas, la structure d'accès (structure à seuil) comprend tous les sous-ensembles possibles de ***k*** éléments parmi ***n***.

Le problème d'implémenter des schémas de partage de secret pour des structures à seuil a été résolu indépendamment par Blakley [94] et de Shamir [95] en 1979.

Les schémas SSS les plus connus dans la littérature incluent le partage de Shamir [95] basé sur l'interpolation de polynômes dans un corps fini, celui de Barkley [94] basé sur la géométrie hyper plane et celui d'Asmuth-Bloom [96] basé sur le théorème des restes chinois [53,60].

III.5.1 Partage de secret à seuil (k,n) de Shamir

Le schéma de Shamir qui utilise l'interpolation de Lagrange, se base sur le fait qu'il existe un seul polynôme de degré (**k-1**) qui passe par **k** points distinct donnés.

Il existe deux cas de figure selon que les parts de secret sont générés et distribués par une seule entité : le distributeur de confiance (trusted dealer), ou par calcul MPC impliquant les participants uniquement (sans dealer). Dans ce paragraphe, nous allons présenter le premier cas, le second cas sera traité au chapitre suivant. Supposons donc l'existence d'un centre de confiance qui met en place le système et distribue les ***n*** parts du secret à ***n*** autorités.

On suppose donc qu'on a ***n*** autorités à qui il faut partager un secret s, et que la coopération d'au moins ***k*** d'entre elles permettent de reconstituer ce secret. Par ailleurs on dispose du dealer qui va faire le partage. L'algorithme est le suivant :

1. Le dealer choisit un nombre premier $p > \max(n, s)$ et pose $a_0 = s$.
2. Le dealer tire au hasard et indépendamment des nombres $a_1, \cdots, a_{k-1} \in$ à $\{0, 1 \cdots, p-1\}$, obtenant ainsi un polynôme :

$$P(z) = a_0 + a_1 z + \cdots + a_{k-1} z^{k-1} \qquad \textbf{(III.6)}$$

3. Le dealer, pour chaque point $z_i \in \{1, \cdots, p-1\}$, où z_i est l'identifiant unique d'un nœud, calcule dans Z_p le secret partiel :

$s_i = P(z_i)$. Il communique secrètement (z_i, s_i) à l'autorité A_i (i=1, …,***n***)

Toute collaboration d'au moins ***k*** autorités permet de reconstituer par interpolation de Lagrange le polynôme P(z), puis de calculer s = P(0). Si au plus ***k − 1*** secrets partiels sont connus, alors tout élément de {0, 1, · · · , p − 1} a une égale probabilité d'être le secret s.

III.6 Partage de fonction à seuil

Le partage de fonction traite le problème de la distribution du calcul d'une fonction (tels que le chiffrement, le déchiffrement ou la signature) parmi plusieurs parties. Les valeurs nécessaires pour le calcul sont distribuées aux participants en utilisant un schéma de partage de secret (SSS). Plusieurs schémas de partage de fonction sont proposés dans la littérature, avec la plupart d'entre elles utilisant le partage de Shamir comme schéma SSS sous-jacent. Dans cette thèse, rappelons-le nous nous intéressons particulièrement au partage de la fonction de signature. La section suivante présente quelques schémas de signature à seuil.

III.6.1 Schémas de signature à seuil

Dans un schéma de signature à seuil, les signatures digitales peuvent être produites par un groupe de nœuds plutôt que par un seul nœud. Contrairement à la signature régulière où le signataire est une entité simple qui tient la clé secrète, dans la signature à seuil la clé secrète est partagée par un groupe de ***n*** nœuds. Afin de produire une signature valide sur un message donné m, les différents nœuds produisent leurs signatures partielles sur ce message, et puis les combinent dans une signature globale. Un schéma distribué de signature réalise un seuil ***k*** < ***n***, si aucune coalition de ***k-1*** nœuds (ou moins) ne peut produire une nouvelle signature valide, même après que le système a produit beaucoup de signatures sur différents messages. Une signature résultant d'un schéma de signature à seuil est identique comme si elle a été produite par un signataire simple possédant la pleine clé secrète de signature. En particulier, la validité de cette signature peut être vérifiée par n'importe quel nœud connaissant la clé publique de vérification correspondante. En d'autres termes, le fait que la signature a été produite d'une façon distribuée est transparent au destinataire de la signature.

Les signataires s'engageront sous une certaine forme de communication qui leur permettra de calculer une signature pour m. Un tel protocole ne devrait divulguer aucune information au delà d'une telle signature. Afin d'obtenir également la propriété de robustesse, de tels protocoles devraient correctement calculer la signature même si jusqu'à ***k-1*** nœuds sont corrompus et se comporter de quelque façon pendant le protocole. Si possible, le calcul exigé par un nœud pour signer dans cette façon distribuée devrait être comparable à l'effort exigé pour une signature ordinaire. L'interaction devrait être réduite à un minimum.

Un schéma de signature à seuil est dit robuste sinon seulement ***k-1*** nœuds (ou moins) ne peuvent pas produire une signature valide, mais également ne peuvent pas empêcher les nœuds de produire une signature à seuil quelconque.
La sécurité de la plupart des schémas de signature est basée sur l'intractabilité d'un des problèmes suivants :

- le problème du logarithme discret dans un groupe multiplicatif (DLP),
- le problème du logarithme discret sur les courbes elliptiques (Elliptic Curve Discrete Logarithm Problem, ECDLP)
- le problème de factorisation en nombres premiers (FP).

Pour le cas des schémas de signature basés sur le problème DLP, la signature robuste à seuil de type EL Gamal ou Schnorr et des schémas de signature de DSA peuvent être trouvés dans [96-98]. Ceux basés sur le problème ECDLP peuvent être trouvés dans [99-101]. A noter que tous ces schémas utilisent les résultats fondamentaux de Feldman [102] et Pedersen [103] pour partager le secret d'une façon distribuée (sans dealer).

Cependant, dans certains cas la solution de dealer est la meilleure qui se peut faire. Par exemple, si le schéma fondamental de signature est RSA (problème FP), on ne sait pas produire une clé partagée sans l'utilisation d'un dealer. RSA s'est avérée être même moins favorable à la construction de schémas robustes [104]. Une solution quelque peu inefficace (exige beaucoup plus de calcul et beaucoup d'interaction entre les signataires) peut être trouvée dans [105].

III.7 Conclusion

Dans ce chapitre, nous avons examiné les modèles de sécurité basés sur la cryptographie asymétriques qui ont été proposés pour fournir la base de confiance et l'authentification dans le contexte des réseaux MANETs. Il est tout à fait clair que le modèle de PKI totalement distribuée représente le modèle le plus approprié au cadre de la solution envisagée dans cette thèse. Ceci nous a amené à introduire les principes de bases et les outils de la cryptographie à seuil sur laquelle repose le modèle en question. Parmi les trois problèmes cryptographiques réputés difficiles et formant les bases des systèmes cryptographiques à seuil, nous avons dirigé notre choix sur le problème DLP en raison de la disponibilité et la complexité relativement moyenne des algorithmes associés. Le chapitre suivant donne la description et la spécification détaillées de notre solution d'auto-configuration pour les réseaux MANETs autonomes, et fournit la preuve de sa sécurité et de sa robustesse.

CHAPITRE 4

Protocole TCSAP Description et Spécification

IV.1 Introduction

Ce chapitre a pour but de concevoir un nouveau protocole d'auto-configuration sécurisé et robuste, remplissant toutes les exigences mentionnées dans le chapitre 2. Pour se faire, nous avons besoin d'un modèle de confiance et d'un schéma d'auto-configuration qui collaboreront à fournir un service d'assignation d'adresses IP, de la sécurité et de la robustesse. Il est aujourd'hui évident que la conception de n'importe quel schéma de service pour les réseaux ad hoc mobiles dans un environnement hostile doit éviter l'utilisation de toute entité centralisée. Un tel serveur centralisé devrait constituer une cible attrayante pour les attaquants. Afin d'être protégé contre toute entité unique, nous adoptons dans notre proposition, pour les deux schémas (de confiance et d'auto-configuration), une approche totalement distribuée basée sur le concept de seuil.

Le modèle de confiance adopté est basé sur l'émulation d'une infrastructure à clé publique en ligne. L'Autorité de Certification en ligne (CA on-line) est totalement distribuée entre tous les nœuds du réseau MANET [59]. Pour la signature des certificats, nous avons adopté un schéma de signature à seuil de type El-Gamal [105]. Le schéma d'Auto-configuration utilisé s'appui sur le protocole MANETconf [7] avec plusieurs modifications permettant l'implémentation d'un service à seuil. Avec cette approche, nous garantissons la disponibilité des services de la certification et de l'auto-configuration.

En plus de l'authentification mutuelle, un nouveau concept de certificat conjointe liant l'adresse IP à la clé publique est introduit pour contrecarrer irrévocablement l'attaque d'usurpation d'adresse IP et l'attaque sybile. Le nouveau protocole conçu est nommé TCSAP (Threshold Cryptography based Secure Auto-configuration Protocol), il fournit à la fois la sécurité et la robustesse et surmonte toutes les limites des approches proposées précédemment, tout en assurant un service efficace d'allocation dynamique d'adresses IP.

Les deux modèles de confiance et d'auto-configuration sont implémentés dans des modules séparés, qui sont respectivement le module DPKI (Distributed Public Key Infrastructure) et le module TCSAP.

Dans la suite de ce chapitre, nous fournirons une description de ces deux modules, ainsi que leurs interactions. Nous nous mettons pour cela dans un contexte de réseau MANET isolé (déconnecté de l'Internet) et spontané (la naissance du réseau repose uniquement sur les nœuds créateurs et aucune partie tierce n'est nécessaire, autrement dit sans dealer).

IV.2 Modèle de confiance à seuil: Module DPKI

Le modèle d'une CA on-line totalement distribuée [59] est donc adopté, où les certificats ne sont pas signés individuellement par les nœuds mais par une coalition de nœuds ; la clé privée ne sera donc jamais reconstituée. Ce modèle s'appui sur des schémas de partage de secret et de signature à seuil.
Dans cette section nous décrivons quelques outils existants nécessaires pour mettre en application ce modèle dans un contexte de réseau MANET isolé et spontané.

Les éléments dont nous avons besoin sont :

- Un schéma de partage de secret (SSS) à seuil; pour cette fin nous utiliserons le partage basé sur l'interpolation de Lagrange de Shamir [95], (voir IV.2.2).

- Produire aléatoirement et d'une façon distribuée (sans partie de confiance) une paire de clés privée (s) et publique (h) du réseau MANET, partager la clé privée du réseau et permettre aux nœuds acteurs de vérifier l'exactitude de leurs parts. Nous utiliserons pour cela un protocole distribué et vérifiable de partage de secret aléatoire. Celui-ci est basé sur le protocole de Pederson (VSS) [103], (voir IV.2.3).

- Attribuer à tout nouveau nœud légitime joignant le réseau MANET une part de la clé privée du réseau. Ceci sera réalisé avec l'appui du protocole proposé par Hong [59], (voir IV.2.4).

- Générer des signatures à seuil pour les certificats délivrés, renouvelés ou révoqués. Nous utiliserons le schéma de signature à seuil de type ElGamal (voir IV.2.5).

IV.2.1 Environnement cryptographique

Soit $P = \{P_1, P_2, \dots, P_n\}$ un ensemble de $\boldsymbol{n}$ nœuds ($|P| = \boldsymbol{n}$) formant un réseau MANET. Chaque nœud P_i a une identité unique u_i. Nous dénotons (s, h) la paire de clés privée et publique du CA on-line totalement distribuée.

La sécurité du modèle proposé est basée sur le problème du logarithme discret (DLP).
Ceci étant, soient *p* et *q* deux nombres premiers très grands tels que *q* divise *p-1*, et soit *g* un élément d'ordre *q* (i-e. : $g \neq 1 \in Z_p^*$ et g^q=1(mod*q*)).
La clé privée $s \in Z_q^*$ et la clé publique correspondante *h* sont liées par : $h = g^s (mod p)$.
(*p*,*q*,*g*,*h*) sont publiques et s est secrète.

IV.2.2 Partage de secret de Shamir

Conformément à [95], un schéma de partage de secret à seuil (***k, n***) nous permet de partager un secret s en $\boldsymbol{n}$ parts de secret s_i ($i \in \{1, .., n\}$) entre $\boldsymbol{n}$ nœuds de sorte que tout sous-ensemble de $\boldsymbol{k}$ nœuds ou plus peuvent coopérer pour reconstituer le secret s et que moins de $\boldsymbol{k}$ nœuds ne peuvent obtenir aucune information sur le secret s. Les parts de secret s_i représentent en réalité des clés partielles qui seront utilisées par les nœuds pour générer des signatures à seuil.

- Initialisation :
  Le dealer choisit aléatoirement un polynôme $f(z)$ de degré $\boldsymbol{k-1}$ dans Z_q

$$f(z) = s + \sum_{l=1}^{k-1} a_l z^l \quad (mod q) \qquad \textbf{(IV.1)}$$

  qui satisfait à : $f(0) = s$.

- Distribution :
  Le dealer calcule les parts de secret s_i par :

$$s_i = f(u_i) = s + \sum_{l=1}^{k-1} a_l {u_i}^l \quad (mod q) \qquad \textbf{(IV.2)}$$

  et les distribue ensuite aux nœuds participants $P_1, P_2, \dots, P_n$

- Reconstitution :

  Etant donné un sous-ensemble $Q = \{P_{i_1}, P_{i_2}, \dots, P_{i_k}\}$ appartenant à la structure d'accès, où $(i_1 < i_2 < \cdots < i_k) \in \{1, .., n\}$.

  L'interpolation de Lagrange donne :

$$f(z) = \sum_{l=1}^{k} B_l(z).s_{i_l} \ (modq) \quad \textbf{(IV.3)}$$

où: $$B_l(z) = \prod_{j \neq l} \frac{z - u_{i_j}}{u_{i_l} - u_{i_j}} \quad \textbf{(IV.4)}$$

  Le secret peut être reconstitué par :

$$s = f(0) = \sum_{l=1}^{k} B_l(0).s_{i_l} \quad \textbf{(IV.5)}$$

IV.2.3 Protocole distribué et vérifiable de partage de secret aléatoire

Le schéma précédent s'appui sur un distributeur de confiance (trusted dealer). Pour notre modèle de confiance, nous avons besoin d'un schéma sans dealer, donc d'un protocole MPC pour effectuer les opérations suivantes :

- Génération de la paire de clés privée/publique (s, h)
- Distribution des parts de secret
- Vérification de l'exactitude des parts (vérifiabilité)

Nous utilisons à cette fin le protocole « Verifiable Secret Sharing » (VSS) de Pederson [103].Nous modifions légèrement ce protocole de telle manière qu'aucune information ne soit coulée à l'adversaire pour effectuer une attaque biasante (biasing attack). On se place dans l'environnement défini dans (IV.2). L'´etablissement des données est fait une fois pour toute et les nœuds participants sont d'accord sur ces valeurs.

Le protocole VSS comprend les phases suivantes :

- Phase 1. Initialisation :

  Chaque $P_i \in P$ choisit aléatoirement un secret $x_i \in Z_q^*$, et calcule une clé publique partielle $h_i = g^{x_i}(modp)$. Contrairement à [] où cette valeur est diffusée publiquement à tous les nœuds, nous utilisons un canal sécurisé (voir phase 2). La clé publique h du réseau MANET (en d'autre terme celle du CA on-line) est calculée par :

$$h = \prod_{i=1}^{n} h_i \ (modp) \quad \textbf{(IV.6)}$$

de sorte que la clé privée s est donnée par :

$$s = \sum_{i=1}^{n} x_i \ \ (modq) \quad \textbf{(IV.7)}$$

- Phase 2. Distribution :

  Dans cette étape, les opérations suivantes sont effectuées :

  – la génération des parts du secret :

  Chaque $P_i \in P$ choisit aléatoirement dans Z_q^* un polynôme $f_i(z)$ tel que :

$$f_i(z) = x_i + \sum_{l=1}^{k-1} a_{il}\, z^l \ \ (modq) \quad \textbf{(IV.8)}$$

  Pour tout $j \in \{1,2,..,n\}$, P_i calcule les parts s_{ij} par :

$$s_{ij} = f_i(u_j)\ (modq) \quad \textbf{(IV.9)}$$

  – la distribution des parts du secret et des clés publiques partielles :

  Chaque P_i transmet en unicast au nœud P_j un message chiffré et signé, contenant s_{ij} et h_i .

  Pour se faire, P_i et P_j doivent utiliser leurs certificats off-line pour établir un canal sécurisé.

  – broadcast des engagements (commitment) requis pour la vérification des parts.

  Chaque P_i diffuse les valeurs d'engagement :

  $A_{il} = g^{a_{il}}(modq)$ pour $l \in \{1,2,..,k-1\}$.

- Phase 3. Vérification :

Grace à la propriété d'homomorphisme, chaque $P_i \in P$ peut vérifier si la part reçue s_{ji} du nœud P_j est correcte ou non, par :

$$g^{s_{ji}} = \prod_{l=0}^{k-1} \left(A_{jl}\right)^{u_j{}^l} \ \ (modp), \qquad pour\ j \in \{1,2,..,n\} \quad \textbf{(IV.10)}$$

A noter que $A_{j0} = h_j = g^{x_j}\ \ (modp)$.

Si la part reçue est correcte, P_i calcule sa part du secret s_i (clé privée partielle) par :

$$s_i = \sum_{j=1}^{n} s_{ji} \ (modq) \qquad \textbf{(IV.11)}$$

Et signe la clé publique h avec sa clé privée partielle (conformément au schéma de signature à seuil prédéfini, voir IV.2.5).

On vérifie aisément que

$$\sum_{i=1}^{n} s_i \ \ (modq) = \sum_{i=1}^{n} x_i \ \ (modq) = s \qquad \textbf{(IV.12)}$$

Si la part reçue est incorrecte, P_i diffuse une erreur et publie la valeur s_{ij} signée.

De toute évidence, le réseau MANET peut être initialisé pour un ensemble $P = \{P_1, P_2, \dots, P_t\}$ de $\boldsymbol{t}$ nœuds ou $\boldsymbol{t \geq k}$**.** Dans notre approche, sans perdre de généralité nous avons établi un protocole d'initialisation du réseau avec $\boldsymbol{k}$ nœuds.

IV.2.4 Attribution d'une part du secret à un nouveau nœud

Pour réaliser un schéma d'autorité de certification en ligne totalement distribuée, un nouveau nœud joignant le réseau doit pouvoir obtenir une part du secret (une part de la clé privée du réseau). Dans notre proposition, avant d'attribuer une part du secret au nouveau nœud nous proposons de l'authentifier par « son certificat en ligne » que nous définirons plus loin.

Pour se faire, nous adoptons l'algorithme distribué proposé par [59]. Etant donné un sous-ensemble $Q = \{P_{i_1}, P_{i_2}, \dots, P_{i_k}\}$ où $(i_1 < i_2 < \cdots < i_k) \in \{1, .., n\}$, un nouveau nœud P_{new} peut calculer sa part du secret s_{new} par :

$$s_{new} = \sum_{l=1}^{k} B_l(z = u_{new})\, s_{i_l} \qquad \textbf{(IV.13)}$$

Les valeurs $B_l(u_{new})s_{i_l}$ sont calculées, signées et envoyées au nœud P_{new} par chaque nœud $P_{i_l} \in Q$.

Comme $B_l(u_{new})$ sont publiquement connues, on a proposé un schéma brouillant (shuffling scheme) pour garder s_{i_l} secret seulement à son propriétaire P_{i_l}.

Dans la technique shuffling, un nonce aléatoire est échangé entre deux serveurs quelconques de Q.

Le nœud avec le petit u_{i_l} choisit et traite le nonce en tant que négatif tandis que l'autre le traite en tant que positif. En conclusion, chaque serveur aura (***k-1***) nonces qu'il ajoute à la valeur $B_l(u_{new})s_{i_l}$ pour obtenir une contribution brouillée de la part du secret (voir [8] pour plus de détails).

IV.2.5 Signature à seuil de type ElGamal

Etant donné que la sécurité du modèle de confiance adopté dans cette thèse est basée sur le problème du logarithme discret, donc tout schéma de signature à seuil de type Nyberg-Ruepple et ou particulièrement de type ElGamal convient pour générer des certificats d'une manière distribuée. Le schéma proposé par Park et Kurosawa [105] est particulièrement intéressent en raison de son efficacité. Les auteurs ont présenté un schéma de signature à seuil construit sur la base d'une variante de DSA (Digital Signature Algorithm, [88]). DSA est basé sur le schéma de signature ElGamal [87].

IV.2.5.1 Schéma de signature ElGamal

Le système de signature ELGamal a été proposé par ElGamal en 1985 [87]. Il est basé sur l'intractabilité du logarithme discret.

Soit p un nombre premier très grand et soit $g \in Z_p^*$ un générateur. La clé privée $s \in Z_p^*$ et la clé publique correspondante h sont liées par : $h = g^s(modp)$, (p,g,h) sont publiques et s est secrète.
La signature d'un message m est le couple (v, w) tel que :

$$v = g^{\gamma} modp \qquad \textbf{(IV.14)}$$

et w est solution de $\gamma w + vs = m\, mod(p-1)$, soit :

$$w = (m - vs)\gamma^{-1} mod(p-1) \qquad \textbf{(IV.15)}$$

où $\gamma \in Z_{p-1}^*$ est un nombre aléatoire choisi par le signataire.

L'équation de vérification est :

$$g^m = h^v v^w \, modp \quad (g^m = g^{vs} g^{\gamma w} \, modp\,) \qquad \textbf{(IV.16)}$$

IV.2.5.2 Schéma de signature DSA

On se met dans les hypothèses de l'environnement cryptographique défini dans (IV.2.1) :

- p est un grand nombre premier (1024 bits)
- q est un grand nombre premier (160 bits) tel que q/p-1
- g générateur de Z_q^*

- $s \in Z_q^*$ est la clé secrète et $h = g^s(modp)$ est la clé publique

Soit $\text{H}(\cdot)$ une fonction de hachage à 160 bits (eg. SHA-1).

Pour signer un message m, le signataire choisit un nombre aléatoire $\gamma \in Z_q^*$ et calcule :

$$v = g^{\gamma} modp\ (modq) \quad \textbf{(IV.17)}$$

$$w = \gamma^{-1}(\text{H}(m) + vs) modq \quad \textbf{(IV.18)}$$

La signature du message m est le couple (v, w).
Pour vérifier la signature, le destinataire vérifie que $1 \leq v, w \leq q - 1$ et que l'égalité suivante est satisfaite :

$$v = g^{\text{H(m)w}^{-1}} h^{vw^{-1}}\ modp\ (modq) \quad \textbf{(IV.19)}$$

IV.2.5.3 Schéma de signature variante de DSA

On utilise les mêmes notations que (IV.2.5.2) :
Signature : (v, w)

$$v = g^{\gamma} modp\ (modq) \quad \textbf{(IV.20)}$$

$$w = (\text{H}(m)\gamma + vs) modq \quad \textbf{(IV.21)}$$

Vérification :

$$v = g^{\text{wH(m)}^{-1}} h^{-v\text{H(m)}^{-1}}\ modp\ (modq) \quad \textbf{(IV.22)}$$

IV.2.5.4 Schéma de signature à seuil basé sur la variante de DSA

Sans perdre de généralité, soit un sous-ensemble $Q = \{P_1, P_2, \dots, P_k\}$ de nœuds désignés pour produire une signature à seuil. Tout d'abord, ces nœuds doivent collaborer pour générer une valeur aléatoire γ.
On utilise pour cela le protocole distribué et vérifiable de partage de secret aléatoire (IV.2.3).
Chaque $P_i \in Q$ obtient une part γ_i de la valeur aléatoire secrète γ.

La valeur publique de γ est $\Gamma = g^{\gamma} mod p$, ($\sum_{i=1}^{k} \gamma_i \ \ (mod q) = \gamma$) de sorte que :

$$v = \Gamma(mod q) \tag{IV.23}$$

Chaque $P_i \in Q$ génère ensuite une signature partielle :

$$w_i = \big(\text{H}(m)\gamma_i + vs_i\big) mod q \tag{IV.24}$$

La signature à seuil (v, w) est donnée par :

$$v = \Gamma(mod q)$$
$$w = \sum_{i=1}^{k} B_i(0) w_i \ \ (mod q) \tag{IV.25}$$

où w est calculée par l'interpolation de Lagrange ($B_i(z) = \prod_{j \neq i} \frac{z - u_j}{u_i - u_j}$) .

IV.3 Modèle d'auto-configuration à seuil : Module TCSAP

Considérons un réseau MANET autonome (isolé). Nous développons un schéma d'auto-configuration stateful inspiré de MANETconf [7]. L'adresse IP pour un nouveau nœud est assignée par un sous-ensemble $Q \subset$ P d'au moins $\boldsymbol{k}$ nœuds ($|Q| \geq \boldsymbol{k}$). Avec cette intension nous distribuons le service d'auto-configuration à tous les nœuds dans le réseau de telle manière que seulement un nombre seuil de nœuds puisse collaborer à exécuter le service.

Contrairement à MANETconf dans lequel une réponse affirmative de tous les nœuds dans le réseau est nécessaire avant d'assigner n'importe quelle adresse IP disponible à un nouveau nœud joignant le réseau, notre schéma réduit l'overhead de communication et la latence en assignant des adresses IP libres sans demander la permission de n'importe quel autre nœud dans le réseau MANET.

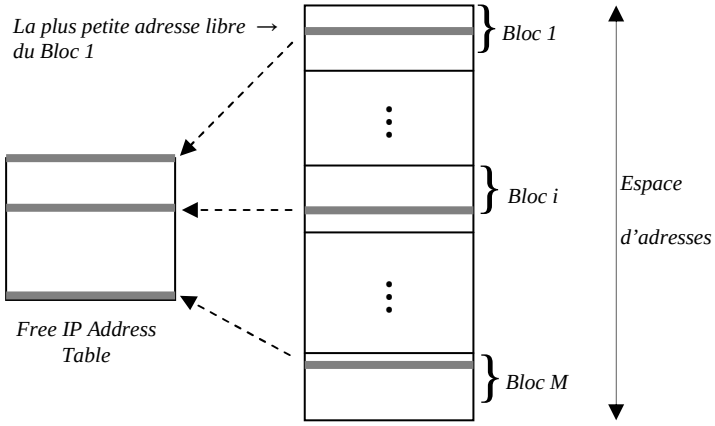


Figure 4.1- Espace d'adressage et 'Free IP Address Table'

Pour réaliser ceci, nous divisons l'espace d'adressage en un certain nombre fixe (dire M) de blocs disjoints d'adresse IP avec des tailles égales (figure 4.1). Nous définissons un bloc d'adresse IP comme une gamme consécutive d'adresses IP. Le paramètre M est une puissance de 2.

IV.3.1 Information d'état maintenue

À n'importe quel instant de la vie du réseau MANET, chaque nœud configuré (ayant obtenu une adresse IP valide, voir IV.6.1) doit maintenir certaines informations d'état définies ci-après :

- ***Free IP Address Table (FAT)*:** contient la plus basse adresse libre de chaque bloc d'adresses. (soit M adresses IP)

- ***Pending IP Address Table (PAT):*** contient les adresses IP récemment assignées qui ne sont pas encore enregistrées**.**

- ***Registered IP Address Table (RAT):*** Chaque entrée dans cette table contient toute adresse IP assignée et enregistrée, l'identité du nœud correspondant, sa clé publique et la période de validité de son certificat on-line (Tableau 4.1). Un nœud enregistré sera supprimé du RAT si son certificat a expiré. Les nœuds souhaitant maintenir leurs adresses IP doivent faire une demande de maintient dans un délai spécifié avant l'échéance de leurs certificats.

- ***Requester Counter (RC):*** ce compteur est maintenu (par tous les nœuds du réseau) pour chaque nouveau nœud demandant une configuration automatique et qu'une adresse IP lui a été assignée mais pas encore enregistrée. Il est incrémenté à chaque nouvelle demande d'auto-configuration. Pour empêcher l'attaque d'épuisement de l'espace d'adressage, les tentatives autorisées pour la demande de service de configuration automatique sont limitées.

Identité du noeud	IP Address	Clé publique	Certificate Validity Period
u_{i_1}			
u_{i_2}			
⋮			
u_{i_n}			

Tableau 4.1 – Table RAT (Registered IP Address Table)

Un nœud configuré doit mettre à jour son information d'état dans les situations suivantes :

- Chaque fois qu'il redémarre.
- Chaque fois qu'il quitte le réseau et le rejoint à nouveau.
- S'il n'a pas été sollicité pendant longtemps pour assurer le service d'auto-configuration.

Le nœud souhaitant mettre à jour son information d'état doit collecter des données redondantes d'au moins $\boldsymbol{k}$ nœuds honnêtes.

IV.3.2 Attribution d'une adresse IP

Un nouveau nœud sollicitant une auto-configuration sera assigné aléatoirement une des plus basses adresses libres contenues dans le ***FAT***, ainsi cela signifie que les adresses IP sont assignées dans un ordre croissant d'un bloc d'adresses IP aléatoirement choisi. Nous imposons au nouveau nœud joignant le réseau d'obtenir son adresse IP d'au moins $\boldsymbol{k}$ nœuds honnêtes. Nous employons à cette fin la signature à seuil décrite ci-dessus et le nouveau concept du certificat commun (voir IV.4).

Après avoir obtenu une adresse IP signée, le nouveau nœud doit diffuser un message signé d'enregistrement à tous les nœuds pour pouvoir participer activement au réseau, c'est-à-dire dans les services de configuration automatique, de routage, etc.

Toute adresse IP assignée qui n'est pas encore enregistrée est supprimée de la table ***FAT*** et maintenue dans la table ***PAT***, par les $\boldsymbol{k}$ nœuds signataires après avoir assigné cette adresse ou par tous les nœuds ayant reçu un message d'enregistrement pour une adresse IP plus haute dans le même bloc d'adresses IP. Si un message d'enregistrement pour une adresse IP est reçu, celle-ci est enlevée de la table ***PAT*** et mise dans la table ***RAT.***

IV.4 Certificat commun de clé publique et d'adresse IP

Nous imposons dans notre approche que chaque adresse IP valide dans le réseau doit être signée par l'autorité de certification en ligne (CA on-line). Pour cela, la coalition des ***k*** nœuds qui ont assigné cette adresse IP devrait produire également une signature à seuil pour cette même adresse. D'autre part, l'adresse IP doit être liée à l'identité du nœud. Pour réaliser ces deux objectives, nous avons deux possibilités :

- Certificats séparés : un certificat pour l'adresse IP et un autre pour la clé publique.
- Certificat commun : dans lequel l'adresse IP et la clé publique sont liés à l'identité du nœud.

Cependant, afin de pouvoir prendre une partie active dans le réseau, un nouveau nœud doit obtenir entre autres :

- Une adresse IP valide
- Un certificat à clé publique signé par la CA on-line
- Une part de la clé privée du réseau

Cela signifie qu'en cas de certificats séparés un nouveau nœud ayant reçu déjà un des deux certificats doit attendre l'autre. En conséquence, il nous semble évident d'opter pour un certificat commun pour des raisons d'efficacité du protocole d'auto-configuration. Nous appelons ce certificat « certificat commun de clé publique et d'adresse IP ».

IV.4.1 Modification du Certificat X.509

La forme standard d'un certificat à clé publique est définie selon la norme X.509 [53,106] (Figure 4.2). Le certificat contient des informations relatives au certificat : son format (version), son numéro de série, la spécification de l'algorithme de chiffrement utilisé pour la signature et sa date de validité. Mais aussi, des informations relatives au porteur : son nom, un identifiant unique, sa clé publique et l'algorithme de chiffrement associé à sa clé publique. Et enfin, des informations relatives à l'autorité de certification : le nom de l'autorité, son identifiant unique et la signature numérique de l'autorité de certification. Une fonction de hachage MD5 ou SHA-1 est appliquée sur les informations contenue dans le certificat. Le résultat est alors chiffré avec la clé privée de l'autorité de certification. On obtient ainsi la signature numérique de l'autorité de certification qui est concaténé aux informations.

Le « certificat commun de clé publique et d'adresse IP » que nous venons d'introduire prendra la même forme avec une légère extension permettant d'ajouter un champ pour insérer l'adresse IP du porteur. Ce champ sera mis par exemple juste après le champ « Identifiant unique du porteur ».

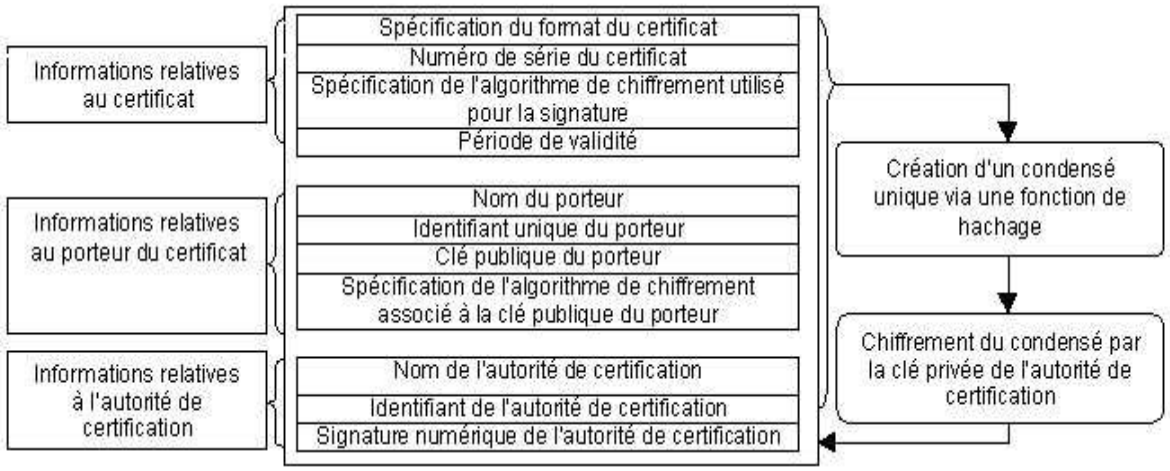


Figure 4.2 – Format d'un certificat X.509

IV.5 Hypothèses

En plus des hypothèses faites sur le contexte de réseaux MANET considéré et le modèle de sécurité utilisé, nous présentons à présent quelques hypothèses relatives au schéma proposé.

H1 :

Dans notre schéma les nœuds peuvent joindre ou quitter le réseau MANET dynamiquement, la seule condition est d'avoir un certificat valide. Nous admettons l'existence d'une autorité de certification hors ligne (off-line) délivrant des certificats à clé publique off-line pour les nœuds légitimes qui participeront dans le réseau MANET. Pour joindre le réseau, chaque nœud doit détenir son « certificat à clé publique off-line» et la clé publique de cette autorité off-line pour être capable de vérifier la validité de n'importe quel « certificat à clé publique off-line» délivré par cette autorité.

Lorsqu'un nœud joint le réseau, deux cas se présentent :

- Si c'est la première fois, le nœud doit utiliser son « certificat à clé publique off-line» pour s'authentifier et obtenir par conséquent un « certificat commun de clé publique et d'adresse IP» pour participer activement au réseau.
- Sinon, nous avons deux sous cas :
  - Si son « certificat commun de clé publique et d'adresse IP» a expiré, il doit agir comme un nœud joignant le réseau pour la première fois.
  - Sinon, il utilisera son « certificat commun de clé publique et d'adresse IP» sous réserve qu'il ne soit pas révoqué.

H2 :

Chaque nœud dans le réseau MANET doit pouvoir vérifier à tout moment si une clé publique est révoquée ou non, par conséquent un schéma de révocation est nécessaire dans le réseau. Les nœuds doivent pouvoir révoquer leurs propres clés publiques ou révoquer les clés publiques de nœuds

malveillants ou compromis. Ceci peut être réalisé par un schéma d'accusation [52] utilisant des signatures à seuil. Nous supposons qu'un tel schéma est mis en application dans le réseau MANET et que chaque nœud maintient au niveau de son module DPKI trois tables contenant respectivement les clés publiques révoquées, les nœuds accusés, et les nœuds malveillants. Nous nous référons à ces tables respectivement comme **RCL** (Revoked Certificates List), **ANL** (Accused Nodes List), et **BL** (Black List). La **RCL** indique ici la liste des différentes révocations de certificat. Chaque révocation doit être signée par l'autorité de certification en ligne (CA on-line). L'**ANL** contient les nœuds qui sont accusés par un certain nombre inférieur au seuil ***k*** de nœuds honnêtes. Si le nombre d'accusations signées par différents nœuds a atteint le seuil, selon la politique de sécurité le nœud accusé sera ou bien considéré en tant que malveillant pour le reste de temps (liste noire, **BL**), ou juste son certificat sera révoqué (**RCL**).

H3 :

Au démarrage du réseau MANET, il n'est pas garanti qu'un nombre suffisant de nœuds soient dans le même voisinage. Nous supposons qu'au moment de l'initialisation du réseau MANET, il y aura dans un même voisinage au moins ***k*** nœuds capables d'initialiser le réseau en utilisant des liens exclusivement locaux (c'est-à-dire des communications à un saut).

H4 :

Il n'est pas garanti que le réseau MANET soit composé d'un nombre suffisant de nœuds à tout instant de la vie du réseau. Nous supposons qu'à tout instant pendant la vie du réseau MANET, il y aura au moins ***k*** nœuds honnêtes (non compromis) pour servir correctement n'importe quel nouveau nœud joignant le réseau.

H5 :

Dans le cas ou le réseau MANET sera déployé pendant de longue période, nous supposons qu'un schéma de partage secret proactif [80] est mis en application pour régénérer périodiquement les parts du secret afin de se défendre des attaques répétées conduites par les adversaires forts et déterminés.

IV.6 Spécification du protocole TCSAP

Cette section spécifie les détails de notre nouveau protocole TCSAP qui implémente le modèle de confiance et le schéma d'auto-configuration à seuil décrits ci-dessus. Nous fournissons une décomposition modulaire du protocole, où chaque composant a une fonctionnalité spécifique. Les propriétés de ces composants fonctionnels et leur interaction définissent le comportement global du protocole (figure 4.3).

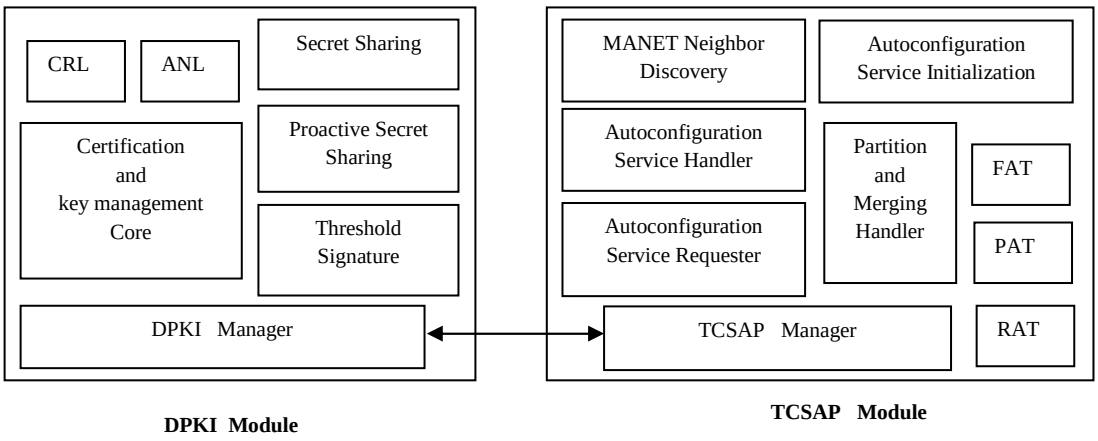


Figure 4.3 - Blocs fonctionnels des modules DPKI et TCSAP

IV.6.1 Etats d'un nœud

Nous définissons dans notre schéma trois états différents pour un nœud participant au réseau MANET :

- Nœud ***Non_configuré*** : tout nœud souhaitant joindre le réseau MANET, et qui n'est pas déjà enregistré avec un « certificat commun de clé publique et d'adresse IP» on-line.

- Nœud ***Configuré*** : tout nœud enregistré dans le réseau MANET détenant :

  - un « certificat commun de clé publique et d'adresse IP» on-line
  - un « certificat à clé publique» off-line
  - la clé publique de l'autorité de certification off-line
  - une part de la clé privée de l'autorité de certification on-line
  - la clé publique de de l'autorité de certification on-line
  - les tables ***FAT***, ***PAT*** et ***RAT***
  - les listes ***RCL***, ***ANL*** et ***BL***

  De tels nœuds devraient pouvoir participer activement au réseau.

- Nœud ***en cours de configuration***: tout nœud ***Non_configuré*** qui a lancé un processus de d'auto-configuration et qui n'a pas encore obtenu de « certificat commun de clé publique et d'adresse IP» on-line pour s'enregistrer.

IV.6.2 Format générique des paquets TCSAP

TCSAP est défini comme un protocole de niveau L 3, il est transporté par IP. Le format général du paquet TCSAP est représenté à la figure 4.4, il comprend les champs suivants :

- TCSAP_TYPE : désigne le type du message TCSAP.
- LENGTH : donne la longueur totale du paquet TCSAP.
- TCSAP_PAYLOAD : ce champ peut contenir les données correspondantes au type du message TCSAP.

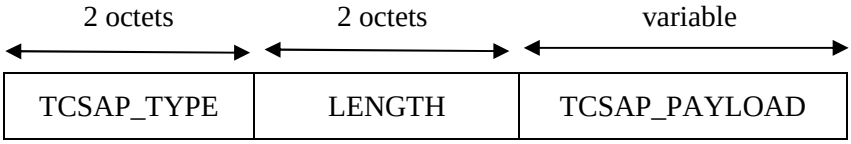


Figure 4.4 – Format générique des paquets TCSAP

IV.6.3 Protocole de Découverte des voisins

Le protocole de découverte de voisins MANET_NDP (MANET Neighbors Discovery protocol) permet à un nœud de découvrir ses voisins d'un saut en diffusant localement et périodiquement des messages de découverte ***Discovery_Request*** signés. La signature ici est faite grâce au certificat à clé publique off-line du nœud.

IV.6.3.1 Format des messages MANET_NDP

Nous avons défini les trois messages suivants pour le protocole sous-djacent MANET_NDP :

- ***Discovery_Request*** : ce message est utilisé par un nœud ***Non_configuré*** dans le protocole de *découverte de voisins* MANET_NDP pour découvrir ses voisins d'un saut. Son format est représenté à la figure 4.5 :

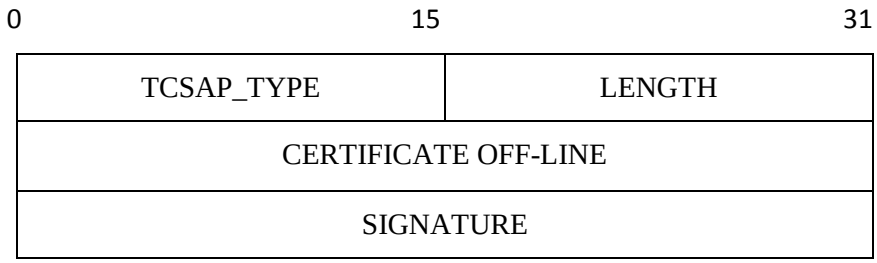


Figure 4.5 – Format du paquet Discovery_Request

TCSAP_TYPE : 0x0101
CERTIFICATE OFF-LINE : certificat off-line du nœud ***Non_configuré***
SIGNATURE : signature du message avec la clé privée du nœud ***Non_configuré***

- ***Discovery_Reply*** : est un message de réponse à tout message ***Discovery_Request*** quand le récepteur est dans l'état ***Non_configuré***. La figure 4.6 donne le format du message :

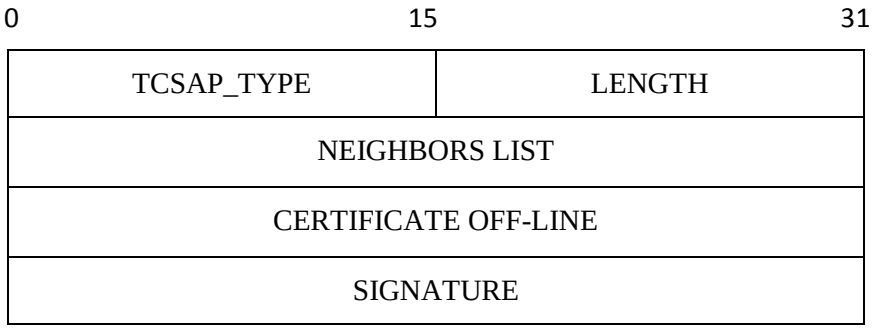


Figure 4.6 – Format du paquet Discovery_Reply

TCSAP_TYPE : `0x0103`
NEIGHBORS LIST : liste des voisins
CERTIFICATE OFF-LINE : certificat off-line du nœud répondant
SIGNATURE : signature du message avec la clé privée du nœud répondant

- ***Discovery_Welcome*** : est un message de réponse à tout message ***Discovery_Request*** quand le récepteur est dans l'état ***Configuré***. La figure 4.7 donne le format du message :

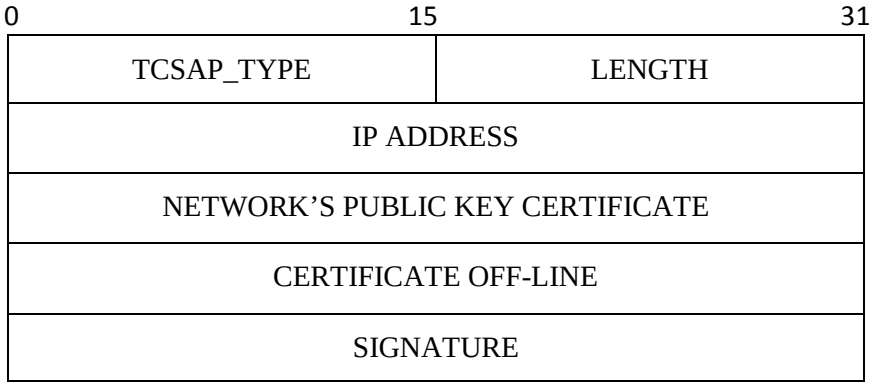


Figure 4.7 – Format du paquet Discovery_Welcome

TCSAP_TYPE : `0x0102`
IP ADDRESS : adresse IP du nœud répondant
NETWORK'S PUBLIC KEY CERTIFICATE : certificat on-line du réseau
CERTIFICATE OFF-LINE : certificat off-line du nœud répondant
SIGNATURE : signature du message avec la clé privée du nœud répondant

IV.6.3.2 Temporisateur

- ***DiscoveryTimer*** : le nœud emploie ce temporisateur pour détecter la présence de ses voisins d'un saut, il permet par conséquent de déterminer un nombre seuil de nœuds fondateurs (pour l'initialisation du réseau). Ce temporisateur est armé chaque fois qu'un message ***Discovery_Request*** est émis. Sa valeur est donnée par DREQ_TIMEOUT.

IV.6.3.3 Cache

- ***Neighbors_Cache*** : le nœud maintient ce cache pour enregistrer ses voisins à un saut. Chaque entrée de ce cache contient l'identité du voisin et l'ensemble de ses voisins.

IV.6.3.4 Opérations du protocole MANET_NDP

Le protocole de découverte de voisins MANET_NDP est exécuté automatiquement par un nœud sur démarrage/redémarrage quand son état est **Non_configuré**. Le nœud diffuse un message ***Discovery_Request*** qui doit contenir le certificat à clé publique off-line et sa signature. A l'émission de ce message, le temporisateur ***DiscoveryTimer*** est armé à la valeur DREQ_TIMEOUT (1000 ms par défaut). Si le temporisateur expire sans que le nœud ne reçoive aucun message ***Discovery_Replay*** ou ***Discovery_Welcome***, le nœud recommence le processus de découverte de voisins MANET_NDP, sinon il agit en fonction du message reçu. Celui-ci dépend de l'état du nœud récepteur du message ***Discovery_Request*** :

- 1er Cas : état **Non_configuré** :

Le nœud récepteur (soit P_i) vérifie la signature, si elle est valide il enregistre l'identité du nœud émetteur et son certificat off-line dans la liste $(N_L)_i$ des voisins à un saut (one-hop Neighbors List) et répond par un message ***Discovery_Reply*** contenant son certificat à clé publique off-line et sa signature, autrement il ignore le message.

Dans ce processus, lorsqu'un nœud reçoit un message ***Discovery_Reply***, il enregistre le voisin répondant P_i ainsi que la liste de ses voisins $(N_L)_i$ dans son cache ***Neighbors_Cache***.

A l'expiration du temporisateur ***DiscoveryTimer***, un nœud détermine l'intersection F_N des listes de ses voisins à un saut par :

$$F_N = \bigcap\nolimits_{P_i \in H} (N_L)_i \qquad \textbf{(IV.26)}$$

où H est l'ensemble de tous les voisins à un saut y compris le nœud lui-même.

Si $|F_N| \geq \boldsymbol{k}$, alors le nœud exécute le protocole d'initialisation MANET_BSP (MANET BootStrapping Protocol), autrement il efface son cache ***Neighbors_Cache*** et recommence le processus de découverte de voisins MANET_NDP.

- 2ème Cas : état **Configuré** :

Le nœud récepteur du message ***Discovery_Request*** vérifie la signature ; si elle est valide, il répond par un message ***Discovery_Welcome*** contenant son adresse IP, son certificat à clé publique off-line et sa signature, autrement il ignore le message. Quand le demandeur reçoit le message de ***Discovery_Welcome***, il conclut qu'un réseau MANET est déjà établi et doit entamer la procédure de demande de configuration automatique par le protocole NODE_ACP (NODE Auto-Configuration Protocol).

- 3ème Cas : état **Configuration en cours** :

Dans cet état, le nœud rejettera tous les messages du processus de découverte de voisins MANET_NDP.

L'algorithme pour le protocole de découverte de voisins MANET_NDP, utilisant la notation « Abstract Protocol Notation » [112], est donné en figure 4.8.

IV.6.4 Protocole d'initialisation du réseau MANET

Nous initialisons le réseau MANET par au moins ***k*** nœuds voisin appelés nœuds fondateurs. Quand un nœud ***Non_configuré*** a découvert un nombre supérieur ou égal à ***k*** voisins ($|F_N| \geq \boldsymbol{k}$) appartenant au même voisinage (figure 4.9), il déclenche directement le protocole d'initialisation MANET_BSP (MANET BootStrapping Protocol). Les nœuds fondateurs doivent conjointement exécuter ce protocole.

```
1 :  bool ready = true;
2 :  ready ∧ nodestate = unconfigured →
3 :     broadcast Discovery_Request ;
4 :     start DiscoveryTimer;
5 :     ready := false;
6 :  receive Discovery_Request →
7 :     if (SigCheck(Discovery_Request )) then
8 :        send Discovery_Replay;
9 :     else
10 :       discard message;
11 :    fi
12 :  receive Discovery_Replay →
13 :     if (SigCheck(Discovery_Replay )) then
14 :        save neighbor's data in cache;
15 :     else
16 :        discard message;
17 :     fi
18 :  receive Discovery_Welcome →
19 :     if (SigCheck(Discovery_Welcome)) then
20 :        stop DiscoveryTimer;
21 :        nodestate := configuration in progress;
22 :        ready := true;
23 :        start Autoconfiguration Service Requesting;
24 :     else
25 :        discard message;
26 :     fi
27 :  receive Init_Start →
28 :     if (SigCheck(Init_Start)) then
29 :        stop DiscoveryTimer;
30 :        nodestate := configuration in progress;
31 :        ready := true;
32 :        process Autoconfiguration Service Initialization;
33 :     else
34 :        discard message;
35 :     fi
36 :  timeout (DiscoveryTimer) →
37 :    if ( |F_N| ≥ k ) then
38 :        nodestate := configuration in progress;
39 :        start Autoconfiguration Service Initialization;
40 :     else
41 :        erase cache;
42 :        nodestate := unconfigured;
43 :        ready := true;
44 :     fi
```

Figure 4.8 – Algorithme pour le protocole de découverte de voisins MANET_NDP

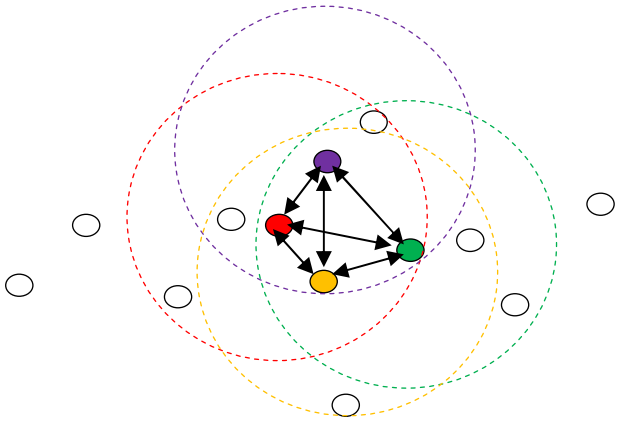


Figure 4.9 – Initialisation d'un réseau MANET avec $k = 4$

IV.6.4.1 Format des messages MANET_BSP

Les messages suivants sont définis pour les opérations du protocole MNET_BSP :

- ***Init_Start*** : le premier nœud qui détecte un nombre suffisant de nœuds dans le même voisinage conformément à la sous-section IV.6.3.4 (2ème cas), diffuse ce message aux autres nœuds fondateurs pour déclencher le processus d'initialisation du réseau MANET. La figure 4.10 donne le format du message ***Init_Start*** :

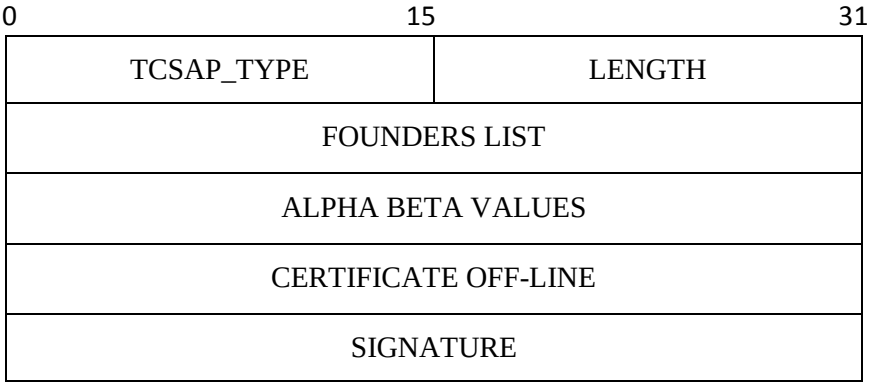


Figure 4.10 – Format du paquet Init_Start

TCSAP_TYPE : `0x0201`
FOUNDERS LIST: liste des nœuds fondateurs
ALPHA BETA VALUES : les valeurs aléatoires α et β choisies par le nœud fondateur
CERTIFICATE OFF-LINE : certificat off-line du nœud fondateur
SIGNATURE : signature du message avec la clé privée du nœud fondateur

- ***Init_Ack*** : ce message est émis par un nœud fondateur comme réponse et accusé de réception au message ***Init_Start***. Le message ***Init_Ack*** utilise le même format que le message ***Init_Start*** avec un TCSAP_TYPE = `0x0202.`

- ***Init_Advert*** : est un message d'annonce employé pour informer tous les nœuds autres que les nœuds fondateurs qu'une initialisation de réseau est entamée (figure 4.11).

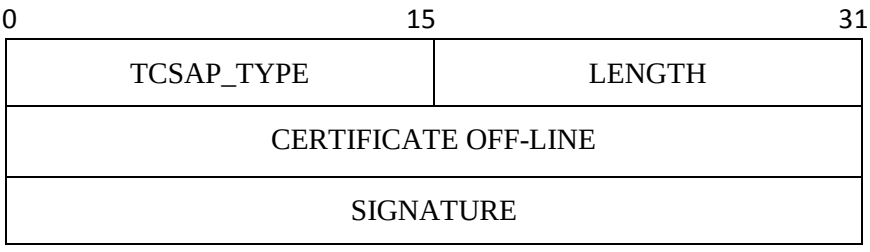


Figure 4.11 – Format du paquet Init_Advert

TCSAP_TYPE : `0x0203`
CERTIFICATE OFF-LINE : certificat off-line du nœud émetteur de ***Init_Advert***
SIGNATURE : signature du message avec la clé privée du nœud émetteur de ***Init_Advert***

- ***Init_Oppos*** : est un message d'opposition aux messages ***Init_Start*** et ***Init_Advert***, utilisé quand une initialisation du réseau MANET est déjà en cours. Le message ***Init_Oppos*** utilise le même format que le message ***Init_Advert*** avec un TCSAP_TYPE = `0x0204.`

IV.6.4.2 Temporisateur

- ***InitTimer*** : le nœud fondateur emploie ce temporisateur pour collecter toutes les valeurs aléatoires ALPHA ET BETA nécessaires pour le calcul des préfixes. Ce temporisateur est armé à la valeur ISTR_TIMEOUT (500 ms par défaut).

IV.6.4.3 Opérations du protocole MANET_BSP

Notre schéma d'auto-configuration est basé sur l'adressage IPv6 [43]. Puisque nous visons un réseau MANET autonome, nous emploierons les adresses « site-local », (figure 4.12).

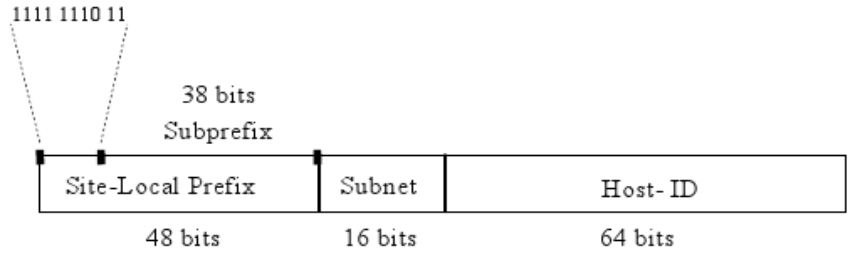


Figure 4.12 – Format d'adresse IPv6 de type « site-local »

La partie Host-ID de 64 bits permet un nombre supérieur à $18x10^{18}$ d'adresses IP possibles, ce qui est suffisant même pour des réseaux MANETs de longue vie et grande échelle.

Le préfixe « site-lcal » est une séquence binaire commençant par FEC, FED, FEE or FEF (dans la notation hexadécimale), c'est-à-dire des préfixes de FEC : : /48 jusqu'à FEF : : /48. Ainsi, le préfixe commence toujours par les dix bits suivants « 1111 1110 11 » ; nous appelons « Subprefix » les 38 bits restant du préfixe.
Le protocole MANET_BSP d'initialisation du réseau MANET est effectué en trois phases :
(L'algorithme pour le protocole MANET_BSP est donné en figure 4.14).

- **Phase 1 : Choix du préfixe du réseau et attribution d'adresses IP aux nœuds fondateurs**

  Dans cette phase, d'abord les nœuds voisins fondateurs doivent choisir le « Subprefix » (38 bits) et le sous-réseau (subnet, 16 bits). Puis, ils doivent configurer leur Host-ID (64 bits) respectifs.

  a) Choix du subprefix et du subnet

  Posons $N_{sp} = 2^{38}$ et $N_{sn} = 2^{16}$. Le premier nœud trouvant le sous-ensemble seuil F_N des voisins fondateurs informe les autres nœuds appartenant à F_N, en leur diffusant un message ***Init_Start*** contenant la liste F_N.

  Chaque $P_i \in F_N$ choisit aléatoirement deux valeurs $\alpha_i \in Z_{N_{sp}}$ et $\beta_i \in Z_{N_{sn}}$, et les diffuse dans un message ***Init_Ack***. Cet échange des valeurs aléatoires α_i et β_i doit se faire dans un temps limité par le temporisateur ***InitTimer***. Après réception de toutes les valeurs α_i et β_i, chaque $P_i \in F_N$ calcule :

$$subprefix = \sum_{P_i \in F_N} \alpha_i \ (mod N_{sp}) \qquad \textbf{(IV.27)}$$

$$subnet = \sum_{P_i \in F_N} \beta_i \ (mod N_{sn}) \qquad \textbf{(IV.28)}$$

Deux initialisations ou plus pourraient être traitées simultanément à différents endroits dans le réseau. Pour éviter ce scénario, nous fournissons la solution suivante :

Juste après avoir reçu la liste des nœuds fondateurs, le nœud avec la plus basse identité annonce un message ***Init_Advert*** pour informer tous les nœuds dans le réseau.

Ce message sera ignoré sauf pour les deux situations suivantes :

- Si le récepteur est dans l'état ***Configuré***, il répond par un message ***Init_Oppos*** indiquant que l'initialisation du réseau est déjà faite.

- Si le récepteur participe à une initialisation de réseau, il arrêtera le processus si la plus basse identité parmi les nœuds fondateurs locaux est plus grande que l'identité du nœud émetteur du message ***Init_Advert***; autrement il continue le processus d'initialisation

b) Auto-configuration des Host-ID

Puisque le préfixe et le subnet sont des valeurs communes pour toutes les adresses IP, seulement l'espace de Host-ID est divisé en M blocs selon (IV.3). En plus de la liste des plus bas Host-ID libre de chaque bloc, la table ***FAT*** contiendra le subnet correspondant (requis pour la gestion du partitionnement et de la fusion de réseau, voir IV.6.6.4). M doit être plus grand que le nombre de nœuds fondateurs ($M > |F_N|$). Les nœuds fondateurs ($P_i \in F_N$) sont classifiés dans un ordre croissant selon leurs identités u_i. Chacun s'auto-assigne la valeur la plus basse dans le bloc d'adresses IP correspondant à son ordre dans cette classification. Ainsi, le nœud avec la plus basse identité u_i obtient la valeur la plus basse du bloc 1, le suivant obtient la valeur la plus basse du bloc 2, et ainsi de suite.

- **Phase 2 : Création du CA on-line**

  Quand la phase 1 est terminée, le manager de TCSAP appelle le manager de DPKI pour commencer le protocole de partage secret après quoi l'autorité de certification distribuée (CA on-line) est crée et sa clé publique est signée (voir IV.2.3 et IV.2.5).

- **Phase 3 : Signature des certificats communs d'adresse IP et de clé publique pour chaque nœud fondateur**

  Dans cette phase, chaque nœud doit obtenir son « certificat commun d'adresse IP et de clé publique » signé par au moins ***k*** nœuds fondateurs. A ce niveau, chaque nœud vérifie l'exactitude du Host-ID avant d'exécuter une signature à seuil. Si ceci se tient, alors le MANET est complètement initialisé et tous les services peuvent être démarrés (configuration automatique, routage, certification en ligne,…).

```
1 : bool ready = true;
2 : nodestate = unconfigured  →
3 :    broadcast  Init_Start ;
4 :    if (my identity is the lowest among F_N) then
5 :        broadcast  Init_Adv;
6 :        nodestate := configuration in progress;
7 :       broadcast  Init_Ack;
8 :       start InitTimer;
9 :       ready := false;
10 :   else
11 :       nodestate := configuration in progress;
12 :   fi
13 : receive Init_Oppos →
14 :   if (SigCheck(Init_Oppos)) then
15 :       nodestate = unconfigured;
16 :       return;
17 :   else
18 :      discard message;
19 :   fi
20 : receive Init_Adertv from  node P_i∈ F_N  Λ ready →
21 :   if (SigCheck(Init_Advert)) then
22 :        if (P_i,s identity is the lowest)  then
23 :           broadcast  Init_Ack;
24 :           nodestate := configuration in progress;
25 :           start InitTimer;
26 :           ready := false;
27 :        else
28 :         continue;
29 :        fi
30 :   else
31 :     discard message;
32 :   fi
33 : receive Init_Advert from  node P_i∉ F_N →
34 :   if (SigCheck(Init_Advert)) then
35 :        if (P_i,s identity < the lowest identity  in F_N)  then
36 :           nodestate = unconfigured;
37 :          return;
38 :        else
39 :          continue;
40 :        fi
41 :   else
42 :     discard message;
43 :   fi
44 : receive Init_Ack from  node P_i∈ F_N →
45 :   if (SigCheck(Init_Adv)) then
46 :      save Init_Ack data from P_i in cache;
47 :   else
48 :      discard message;
49 :    fi
50 : timeout (InitTimer) →
51 :  if (each Init_Ack from P_i∈ F_N  has been deceived)  then
52 :      compute subprefix and subnet;
53 :      self-assign IP Address;
54 :     call DPKI Manager;
55 :     (secret sharing and certificates signing)
56 :   else
57 :      nodestate = unconfigured;
58 :      return;
59 :   fi
```

Figure 4.13 – Algorithme pour le protocole d'initialisation MANET_BSP

IV.6.5 Protocole d'auto-configuration d'un nouveau nœud

Dans le schéma proposé, un nouveau nœud joignant le réseau MANET est assigné une adresse IP au moyen du protocole NODE_ACP (NODE Auto-Configuration Protocol). Ceci doit se faire par l'attribution d'un « certificat commun d'adresse IP et de clé publique » au nouveau nœud et l'enregistrement de ce dernier au près de tous les participants du réseau. Dans cette auto-configuration le nouveau nœud est désigné par ***Requester***, il est aidé par un voisin appelé ***Allocator*** (figure 4.14).

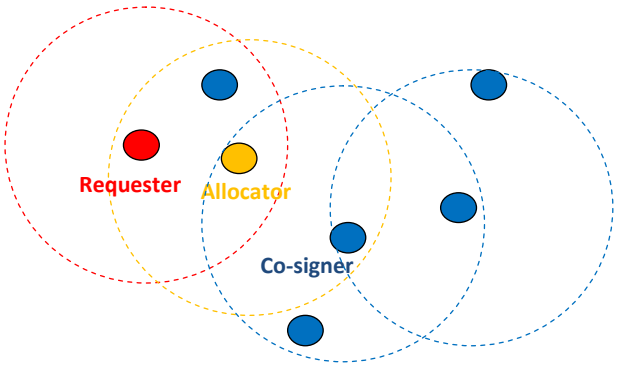


Figure 4.14 - Allocation d'adresse IP (Expanding ring search)

IV.6.5.1 Format des messages NODE_ACP

Nous avons défini pour le protocole NODE_ACP les messages suivants :

- ***Config_Request*** : ce message est utilisé par un nœud dans l'état ***Non_configuré*** pour demander une auto-configuration d'adresse IP suite à la réception du message ***Discovery_Welcome***. La figure 4.15 donne le format du ***Config_Request*** :

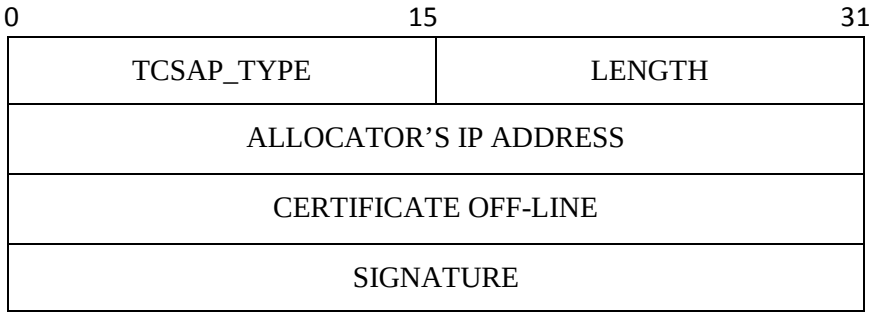


Figure 4.15 – Format du paquet Config_Request

TCSAP_TYPE : `0x0301`
ALLOCATOR'S IP ADDRESS : adresse IP de ***l'Allocator***
CERTIFICATE OFF-LINE : certificat off-line de ***l'Allocator***
SIGNATURE : signature du message avec la clé privée de ***l'Allocator***

- ***Config_Reply*** : ce message est utilisé par un nœud dans l'état ***Configuré*** comme réponse au message ***Config_Request***. Il s'agit d'un message unicast dirigé vers l'***Allocator***. Il comprend la liste des subnets, la liste des plus bas Host-ID libres de chaque bloc (une copie de son ***FAT***), le HopLimit reçu, et enfin la signature (figure 4.16).

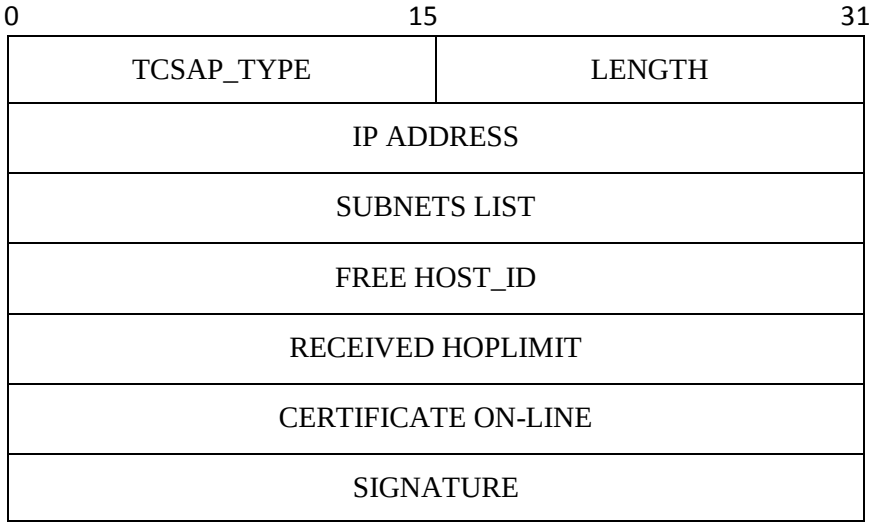


Figure 4.16 – Format du paquet Config_Replay

TCSAP_TYPE : `0x0302`

IP ADDRESS : adresse IP de l'émetteur

SUBNETS LIST : liste des subnets disponibles

FREE HOST_ID : liste des plus bas Host-ID libres de chaque bloc

RECEIVED HOPLIMIT : HopLimit reçu

CERTIFICATE ON-LINE : certificate on-line

SIGNATURE : signature du message avec la clé privée de l'émetteur

- ***Config_Cert_Request*** : ce message est utilisé par l'***Allocato***r pour demander un « certificat commun d'adresse IP et de clé publique » pour le ***Requester*** (figure 4.17).

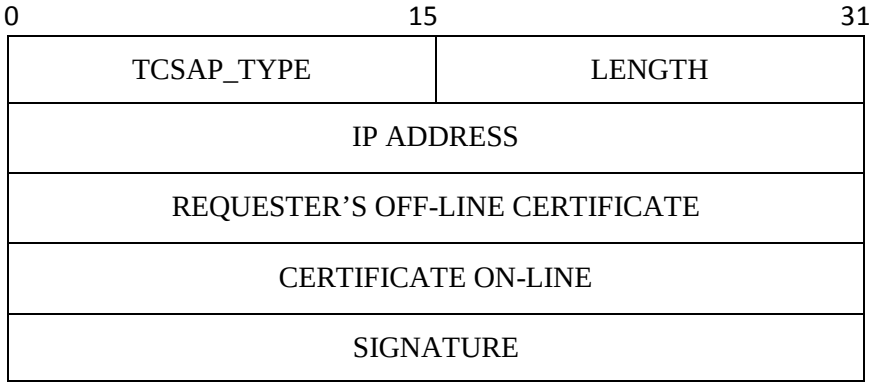


Figure 4.17 – Format du paquet Config_Cert_Request

TCSAP_TYPE : `0x0303`

IP ADDRESS : adresse IP de l'***Allocator.***

REQUESTER'S OFF-LINE CERTIFICATE : certificate off-line du nouveau nœud sollicitant une adresse IP .

CERTIFICATE ON-LINE : certificate on-line de l'***Allocator***.
SIGNATURE : signature du message avec la clé privée de l'***Allocator***.

- ***Config_Cert_Reply*** : ce message est utilisé par le combineur comme réponse au message ***Config_Cert_Request***. Il permet de délivrer le certificat on-line au ***Requester*** par le biais de l'***Allocator*** (figure 4.18).

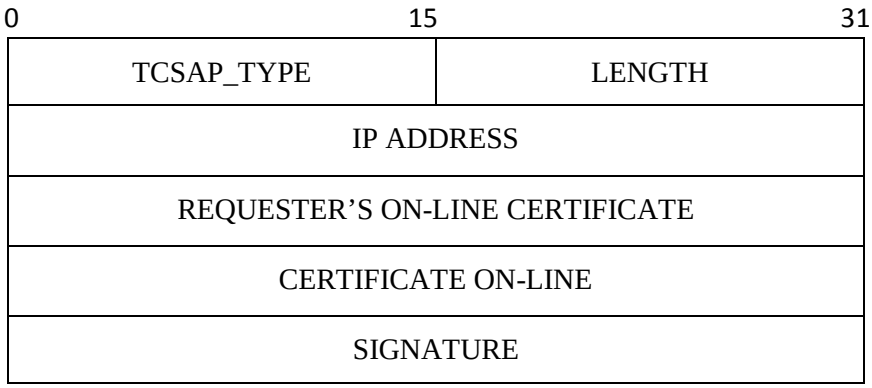


Figure 4.18 – Format du paquet Config_Cert_Replay

TCSAP_TYPE : `0x0304`
IP ADDRESS : adresse IP du combineur.
REQUESTER'S ON-LINE CERTIFICATE : certificat on-line du demandeur.
CERTIFICATE ON-LINE : certificat on-line du combineur.
SIGNATURE : signature du message par le combineur.

- ***Config_Info*** : suite à la réception du message ***Config_Request*** du ***Requester***, l'***Allocator*** émet un message ***Config_Info*** au ***Requester*** pour l'informer du TRANSACTION_ID. Ce message est ensuite émis périodiquement pour pouvoir détecter une migration (figure 4.19).

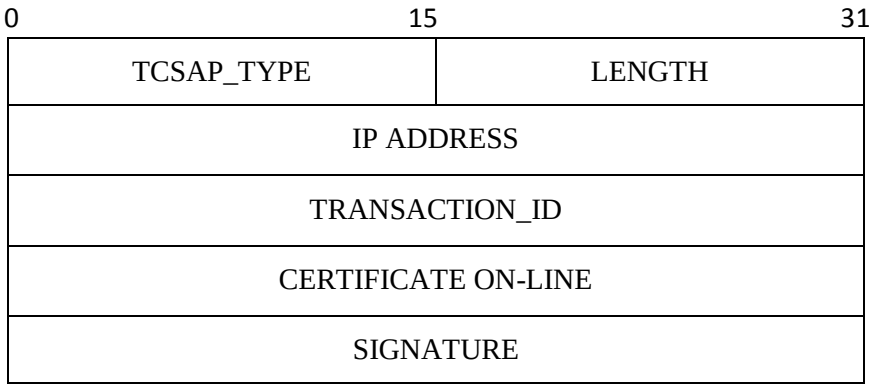


Figure 4.19 – Format du paquet Config_Info

TCSAP_TYPE : `0x0305`
IP ADDRESS : adresse IP de l'***Allocator***.
TRANSACTION_ID : triplet formé de l'identité du nœud
CERTIFICATE ON-LINE :
SIGNATURE :

- ***Config_New_Allocator*** : suite à une migration, ce message est utilisé par le nouveau ***Allocator*** pour contacter l'ancien ***Allocator***.

- ***Config_Old_Allocator*** : ce message est utilisé par l'ancien ***Allocator*** pour répondre au nouveau ***Allocator***.

- ***Config_Advert*** : ce message est utilisé par le combineur pour informer tous les nœuds du réseau qu'une adresse IP a été attribuée à un nouveau nœud.

- ***Config_Alert*** : ce message est utilisé pour annoncer qu'un nœud malveillant a été découvert.

- ***Config_Register*** : ce message permet à un nouveau nœud qui vient d'obtenir un « certificat commun d'adresse IP et de clé publique » de s'enregistrer auprès du réseau MANET.

- ***Config_Error*** : ce message est envoyé comme réponse d'erreur à tout message du protocole NODE_ACP.

IV.6.5.2 Temporisateurs

- ***ConfigTimer*** : un nœud (***Allocato***r) emploie ce temporisateur pour déterminer le nombre de serveurs pour la configuration automatique. Ce temporisateur est armé chaque fois qu'un message ***Config_Request*** est diffusé dans un rayon donné. Sa valeur est donnée par CREQ_TIMEOUT.

- ***ConfigCertTimer*** : l'***Allocato***r utilise ce temporisateur à l'envoi d'un message ***Config_Cert_Request*** pour limiter le délai d'attente de réception d'un message ***Config_Cert_Reply*** du combineur. Sa valeur est donnée par CCREQ_TIMEOUT.

- ***RegistrationTimer*** : ce temporisateur est utilisé par un nœud Configuré dans la procédure d'enregistrement d'un nouveau nœud. Sa valeur est donnée par CREG_TIMEOUT.

IV.6.5.3 Cache

- ***TCSAP_Server*** : un nœud (***Allocato***r) maintient ce cache pour enregistrer les serveurs d'auto-configuration dans un rayon donné.

IV.6.5.4 Opérations du protocole NODE_ACP

Le protocole NODE_ACP est composé de 4 phases : (1) recherche des serveurs, (2) choix des cosignataires, (3) signature à seuil du « certificat commun d'adresse IP et de clé publique » et (4) enregistrement auprès du MANET.

- **Phase 1 : Recherche des serveurs**

Un nouveau nœud qui a reçu un message ***Discovery_Welcome*** valide en réponse à son message ***Discovery_Request*** fait automatiquement une demande d'auto-configuration. L'émetteur du premier message ***Discovery_Welcome*** reçu par ***Requeste***r est choisi comme ***Allocator.***

D'abord, le ***Requester*** s'auto-assigne une adresse « link-local » constituée du préfixe local bien connu FE80 : :0/64 et du Host-ID de l'adresse IP de l'***Allocator***. Puisque ce Host-ID est unique, l'adresse « link-local » obtenue est également unique. Puis, il émet à l'***Allocator*** un message ***Config_Request*** comprenant son certificat à clé publique off-line et sa signature.

L'***Allocator*** à son tour diffuse ce message dans un rayon prédéfini r_k (HopLimit = $r_k \leq k$), et arme le temporisateur ***ConfigTimer*** à la valeur CREQ_TIMEOUT (500 ms par défaut) pour chercher les serveurs dans cette portée. L'algorithme pour cette procédure est donné en figure 4.20.

```
1:  int r_k , HopLimit , numServers;
2:  numServers < k →
3:     self-assign link local IP Address;
4:     HopLimit := r_k;
5:     broadcast Config_Request ;
6:     start ConfigTimer;
7:  receive Config_Replay →
8:     if (SigCheck(Config_Replay)) then
9:        save servers data in cache;
10:    else
11:       discard message;
12:    fi
13: timeout (ConfigTimer) →
14:    if (numServers < k ) then
15:       if (r_k < k) then
16:          erase cache;
17:          r_k := r_k + 1;
18:       else
19:          nodestate = unconfigured;
20:          return;
21:       fi
22:    else
23:       start (coalition selection, certificate requesting and registration);
24:    fi
```

Figure 4.20 – Algorithme de recherche des serveurs

```
1 : const   n, RC_thresh  : integer
2 : int     RC            : array[0..n-1] {initialy RC=0}
3 : receive Config_Request from node P_i →
4 :   if ( P_i ∈ BL) then
5 :       discard message;
6 :   else if SigCheck(Config_Request) then
7 :           if (RC[i] < RC_thresh ) then
8 :              RC[i] := RC[i] + 1;
9 :              send Config_Replay to node P_i ;
10 :          else
11 :             discard message;
12 :             set node P_i as malicious and publish accusation;
13 :          fi
14 :      else
15 :         discard message;
16 :      fi
17 :  fi
```

Figure 4.21 – Algorithme de traitement d'un message Config_Request par un serveur

Tout récepteur du message ***Config_Request*** vérifie si le ***Requester*** n'est pas dans la table ***BL***, puis il vérifie la signature ; si le ***Requester*** existe dans la table ***BL***, ou si la signature est invalide le message est jeté. Autrement, il vérifie le compteur RC (Requester Counter). Si le ***Requester*** a déjà atteint la limite des tentatives autorisées pour l'auto-configuration, il est déclaré malveillant et le message est jeté. Autrement, le destinataire envoie un message ***Config_Reply*** comprenant son « certificat commun d'adresse IP et de clé publique », la liste des subnets, la liste des plus bas Host-ID libres de chaque bloc (une copie de son ***FAT***), le HopLimit reçu, et enfin sa signature.
L'algorithme pour le traitement du message de ***Config_Request*** par un cosignataire est donné en figure 4.21.

A chaque réception d'un message ***Config_Reply***, l'***Allocator*** vérifie la validité du message, regarde dans ses tables ***CRL*** et ***BL*** si aucun membre de la coalition n'est malveillant ni son certificat est révoqué, ensuite il met à jour son cache, et vérifie si le nombre total des messages ***Config_Reply*** reçus (et donc le nombre de serveurs) a atteint le seuil ***k***. Dans le cas positif, il démarre la phase 2 : choix des cosignataires, autrement il ne fait rien.
A l'expiration du temporisateur ***ConfigTimer***, si le nombre total des messages ***Config_Reply*** reçus par l'***Allocator*** est inferieur au seuil ***k***, alors il rediffuse le message en incrémentant le HopLimit. Autrement, il démarre la phase 2.

- **Phase 2 : Choix des cosignataires**

Dans cette phase, l'***Allocator*** choisit parmi les serveurs disponibles un sous-ensemble (coalition) de ***k*** cosignataires et leur demande un « certificat commun d'adresse IP et de clé publique », Le procédé est récapitulé dans les points suivants (l'algorithme est donné en figure 4.22) :

```
1:  bool ready = true;
2:  MaliciousServerTable = ∅;
3:  ready →
4:     select closest servers excluding malicious ones;
5:     select freeHostID;
6:     send Config_Cert_Request to selected servers;
7:     start ConfigCertTimer;
8:     ready := false;
9:  receive Config_Cert_Replay from combiner →
10: if (SigCheck(Config_Cert_Replay)) then
11:  stop ConfigCertTimer;
12:  start registration;
13: else
14:    discard message;
15: fi
16: receive Config_Alert →
17: if (SigCheck(Config_Alert)) then
18:  update MaliciousServerTable;
19:  stop ConfigCertTimer;
20:  ready :=true;
21: else
22:    discard message;
23: fi
24: timeout (ConfigCertTimer) →
25:  nodestate = unconfigured;
26:  return;
```

Figure 4.22 – Algorithme de choix des co-signataires

- l'***Allocator*** sélectionne parmi les serveurs les proches (en fonction du HopLimit figurant dans les messages ***Config_Reply***) une coalition de ***k*** cosignataires.
- Il choisit aléatoirement un plus bas Host-ID libre commun à tous les membres de la coalition choisie.
- Il unicast à ces membres un message ***Config_Cert_Request*** comprenant la liste des membres de la coalition et le plus bas Host-ID libre choisi, et attend la réception d'un message ***Config_Cert_Reply*** du combineur dans un délai déterminé par le temporisateur ***ConfigCertTimer*** armé à la valeur CCREQ_TIMEOUT (1000 ms par défaut).

- **Phase 3 : Signature à seuil du « certificat commun d'adresse IP et de clé publique »**

Chaque membre dans la coalition vérifie la validité du message ***Config_Cert_Request***, regarde dans ses tables ***CRL*** et ***BL*** si aucun membre de la coalition n'est malveillant ni son certificat est révoqué.
Si ceci se tient, alors chaque membre démarre le protocole de signature à seuil du « certificat commun d'adresse IP et de clé publique » pour le ***Requester***.

Le membre de la coalition avec la plus basse adresse IP agira en tant que combineur des signatures partielles, il répond à l'***Allocator*** par un message ***Config_Cert_Reply***, et a en outre la tâche d'informer tous les nœuds du réseau par un message ***Config_Advert*** qu'une adresse IP a été attribuée au nœud en question.

```
1:  receive Config_Cert_Request from Pi →
2:      if (SigCheck(Config_Cert_Request)) then
3:          if (requested IP Address ∈ RAT) then
4:              send Config_Error to Pi ;
5:          else if (there is a malicious node among the
                    selected servers ∈ BL or ∈ CRL)  then
6:              send Config_Alert to Pi and other servers;
7:          else
8:              call DPKI Manager; (certificate issuing)
9:          fi
10:     fi
11:     else
12:         discard message;
13:     fi
```

Figure 4.23 – Traitement d'un message ***Config_Cert_Request*** par un cosignataire

En conséquence, tous les nœuds incrémentent son compteur *RC* et suppriment cette adresse du ***FAT*** et l'enregistrent dans le ***PAT***, ainsi cette adresse ne sera pas choisie pour un autre nouveau nœud.

Si un nœud malveillant a été découvert parmi les membres de la coalition, un message de ***Config_Alert*** est envoyé aux membres honnêtes de la coalition et à l'***Allocator***. Celui-ci effectue alors un nouveau choix de coalition tout en excluant le nœud malveillant. L'algorithme pour ce traitement est en donné en figure 4.23.

- **Phase 4 : Enregistrement auprès du MANET**

Après réception du « certificat commun d'adresse IP et de clé publique », le ***Requester*** vérifie sa validité et procède à l'enregistrement. Cet enregistrement est indispensable pour qu'un nœud soit considéré comme membre du réseau MANET. Pour se faire, le nouveau nœud diffuse un message ***Config_Register*** à tous les nœuds dans le réseau. Le message doit inclure le certificat on-line délivré au nouveau nœud et la signature du paquet IPv6 entier. Ceci affirme que le message a été envoyé à tous les nœuds. Cette demande doit être traitée par chaque nœud sans accusé de réception, l'algorithme pour cet enregistrement est donné en figure 4.24.

```
1:  bool ready = true;
2:  ready →
3:     broadcast Config_Register ;
4:     start RegistrationTimer;
5:     ready := false;
6:  receive Config_Register from node Pi →
7:     if (SigCheck(Config_Register)) then
8:        if (my identity < Pi's identity) then
9:           send Config_Error to Pi ;
10:       else
11:          nodestate := unconfigured;
12:          stop RegistrationTimer;
13:          return;
14:       fi
17:    else
18:       discard message;
19:    fi
20: receive Config_Error from node Pi →
21:    if (SigCheck(Config_Error)) then
22:       if (my identity < Pi's identity) then
23:          send Config_Error to Pi ;
24:       else
25:          nodestate := unconfigured;
26:          stop RegistrationTimer;
27:          return;
28:       fi
29:    else
30:       discard message;
31:    fi
32: timeout ( RegistrationTimer) →
33:     nodestate = configured;
34:    return;
```

Figure 4.24 – Algorithme d'enregistrement d'un nouveau nœud et résolution de conflit d'adresses

```
1:  int LowestID = 0;
2:  receive Config_Register →
3:     if (Config_Register is a broadcast message) then
4:        if (SigCheck(Config_Register)) then
5:           if (the requester IP Address ∈ PAT) then
6:              if (there is a registration in progress for this
                   same IP Adress) then
7:                 LowestID := lowest identity of the
                              requesters for the same IP
8:  Adress;
9:              else
10:                start RegistrationTimer;
11:                LowestID := requester's identity;
12:             fi
13:          else
14:             discard message;
15:          fi
16:       else
17:          discard message;
18:       fi
19:    else
20:       discard message;
21:    fi
22: timeout (RegistrationTimer) →
23:    register the IP Adress in the RAT with LoewstID;
       delete the Ip Adress fron the PAT;
```

Figure 4.25 – Algorithme de traitement d'une demande d'enregistrement

Un conflit peut se produire par l'acquisition d'une même adresse IP quand deux nœuds ou plus essayent de joindre le réseau simultanément et qu'un même Host-ID est choisi. Noter que l'enregistrement d'une même adresse IP peut avoir lieu seulement pendant un intervalle limité de temps. Ce conflit est résolu sur la base des identités des nœuds et le temporisateur RegistrationTimer.

D'une part, le nœud qui a diffusé un message de ***Config_Register*** pour une adresse IP donnée, et qui reçoit d'un autre nœud pendant l'intervalle du temps défini par RegistrationTimer un message ***Config_Register*** pour la même adresse IP doit exécuter un processus de résolution.

Dans ce processus, le nœud récepteur compare son identité à celle de l'expéditeur. S'il a la plus basse identité, il maintient son adresse IP et unicast un message ***Config_Error*** à l'autre nœud. Autrement, il répète le service de configuration automatique où une autre adresse IP sera choisie. L'algorithme pour ce processus est donné en figure 4.24.

D'autre part, chaque nœud dans le réseau MANET, suite à la réception d'un message ***Config_Register***, déclenche RegistrationTimer et retarde l'enregistrement à son expiration. Si un autre message ***Config_Register*** pour la même adresse IP est reçu avant que le temporisateur expire, alors le nœud avec la plus basse identité sera enregistré. L'algorithme correspondant est donné en figure 4.25.

IV.6.6 Scénarios de mobilité et changement de topologie

A cause de la mobilité des nœuds, différents scénarios peuvent se produire au cours de l'existence du réseau MANET, nécessitant des traitements spécifiques.

IV.6.6.1 Migration du *Requester* /*Allocator*

Quand un ***Requester*** s'éloigne de son ***Allocator*** ou vice versa avant l'acquisition d'un certificat on-line (ce déplacement est détecté par l'absence de réception des messages ***Config_Info***), le ***Requester*** choisit un nouveau ***Allocator*** et lui communique l'adresse IP de l'ancien ***Allocator*** et le TRANSACTION_ID. Le nouveau ***Allocator*** envoie alors un message ***Config_New_Allocator*** à l'ancien ***Allocator***. Si l'ancien ***Allocator*** n'a pas quitté le réseau, il met le nouveau ***Allocator*** en cache et répond par un message ***Config_Old_Allocator***. Le certificat on-line du ***Requester*** lui sera délivré par le biais du nouveau ***Allocator***.

IV.6.6.2 Départ de l'*Allocator*

Quant le nouveau ***Allocator*** n'arrive pas à joindre l'ancien ***Allocator***, une nouvelle transaction est créée et un nouveau TRANSACTION_ID est attribué par le nouveau ***Allocator*** au ***Requester***.

IV.6.6.3 Départ d'un nœud

Quand un nœud en état ***Configuré*** part du réseau, il garde son adresse IP. La prochaine fois, s'il souhaite joindre le réseau, il peut utiliser cette adresse tant que son certificat on-line reste valide. Si le

certificat a expiré, il doit demander une nouvelle adresse, et par conséquent un nouveau « certificat commun d'adresse IP et de clé publique » en utilisant à nouveau son certificat off-line.

IV.6.6.4 Partitionnement et fusion d'un réseau MANET

Le réseau MANET peut se diviser en deux ou plusieurs partitions en raison de la mobilité des nœuds. Pour détecter cette division, nous adoptons le protocole décrit dans [7] dans lequel des messages à diffusion générale périodiques sont exécutés par le nœud possédant la plus basse adresse IP. Dans notre schéma, nous définissons une partition comme un sous-ensemble de nœuds avec une taille supérieure ou égal au seuil ***k***, et chaque partition admet une identité unique définie par un quadruplet :

- la plus basse adresse IP en service dans la partition,
- le préfixe de réseau,
- le(s) subnet(s) en service dans la partition,
- la clé publique du réseau.

Quand un groupe de moins de ***k*** nœuds est disloqué du réseau MANET, il ne sera pas considéré comme partition. L'arrivée d'autres nœuds ***Non_configurés*** devraient aider ce groupe à créer, selon le protocole TCSAP, un nouveau réseau indépendant s'ils demeurent isolés du réseau d'origine.

Considérons une division d'un réseau MANET en deux ou plusieurs partitions. L'identité de la partition qui contiendra le nœud avec la plus basse adresse IP restera sans changement. Cette partition continuera à fournir le service de configuration automatique sans aucun changement (c'est-à-dire sans effacement d'adresses ni changement de subnet). Les autres partitions doivent produire de nouvelles identités. Pour se faire, le nœud avec l'adresse IP la plus basse en service dans la partition choisit aléatoirement ***k*** nœuds pour mettre en place un nouveau subnet. Ainsi, la nouvelle identité sera représentée par le quadruplet suivant :

- la plus basse adresse IP en service dans la partition,
- le préfixe de réseau d'origine
- les subnets en service dans la partition (y compris le nouveau)
- la clé publique du réseau d'origine.

Chacune de ces partitions fournira le service de configuration automatique en utilisant uniquement le nouveau subnet. Il n'y aura aucun effacement d'adresses.

Quand une fusion des partitions se produit, une synchronisation de l'information d'état est nécessaire. Le nœud ayant la plus basse adresse IP dans chaque partition rassemble une signature à seuil sur l'information d'état de sa partition et diffuse un message comprenant cette information d'état avec la signature à seuil. Chaque nœud met à jour son information d'état selon ces données, ceci a pour

résultat une information d'état globale unifiée. Le cas où la fusion de deux réseaux indépendant n'est pas vraiment une issue puisque pratiquement il n'y aura aucun conflit d'adresse dû à la probabilité très faible de collision des préfixes de réseaux et des clés publique de réseaux.

IV.7 Analyse de sécurité

Dans ce travail, nous avons adopté une approche cryptographique à seuil (***k***,***n***) avec l'hypothèse $\boldsymbol{n \geq 2k-1}$, pour garantir la sécurité et la robustesse en présence d'un adversaire fort avec au plus $\boldsymbol{k-1}$ nœuds malicieux.

A la différence des approches précédentes, qui se basent sur un seul nœud pour initialiser le réseau ou fournir le service d'auto-configuration, notre schéma ne s'appui sur aucune partie de confiance tierce. Le réseau est initialisé par au moins ***k*** nœuds voisins et le service de configuration automatique est fourni par au moins ***k*** nœuds honnêtes arbitraires. Ainsi, nous évitons toute cible d'attaque unique pour l'adversaire. Ceci est particulièrement intéressant pour les réseaux MANETs spontanés qui sont déployés dans des environnements hostiles.

Notre schéma permet de fournir un service de configuration automatique totalement réparti sur tout le réseau, et un nouveau nœud joignant le réseau n'a besoin d'aucune distribution particulière de ses voisins pour être configuré avec les paramètres de réseau et de sécurité nécessaires. Par conséquent, la disponibilité du service est garantie quelque soit la topologie du réseau.

Le mécanisme d'authentification mutuelle forte (avec les certificats off-line) permet aux serveurs d'authentifier le demandeur (***Requester***), où seuls les nœuds légitimes peuvent participer au réseau, mais également au demandeur d'authentifier les serveurs. Cependant, des nœuds malveillants peuvent être présents parmi les serveurs choisis par l'***Allocator***. Pour cette raison, des messages ***Config_Alert*** sont employés pour isoler ces nœuds malicieux. La vérification de signature à seuil devrait également être employée pour isoler les nœuds ayant un comportement malveillants et qui ne sont pas encore dans la liste ***BL***.

Notons également que l'authenticité de chaque message est garantie par la signature apposée à chaque message. Le message est authentifié et son intégrité est préservée.

Par conséquent, et contrairement aux limitations des diverses approches analysées dans [33], aucune des attaques citées dans le chapitre 2 ne semble mettre en défaut notre proposition. Dans ce qui suit, nous donnons des démonstrations de sécurité vis-à-vis ces attaques :

- **Attaque d'usurpation d'adresse IP**

Hypothèses :

- Chaque nœud légitime admis dans le réseau MANET détient un « certificat commun d'adresse IP et de clé publique ».
- Pour chaque message émis, une signature est apposée par l'émetteur.
- Intractabilité du problème DLP
- L'adversaire ne peut compromettre plus de (***k-1***) nœuds.

Assertion : L'attaque d'usurpation d'adresse IP est infaisable.

Preuve :

Un nœud malveillant qui veut usurper une adresse IP inutilisé ou une adresse IP déjà assignée doit se munir d'un « certificat commun d'adresse IP et de clé publique » dans lequel sa clé publique est liée à l'adresse IP usurpée. Autrement, il sera très facile de détecter l'attaque puisque pour chaque message reçu l'adresse IP de l'émetteur doit être vérifiée avant de procéder à la vérification de la signature apposée au message. L'adresse IP de l'émetteur doit correspondre à celle qui apparaît dans son certificat. Le certificat doit être valide et non révoqué. En conséquence, grâce à ce nouveau mécanisme il est impossible d'effectuer l'attaque d'usurpation d'adresse IP, à moins que l'adversaire peut obtenir la clé privée du réseau, ce qui est équivalent à résoudre le problème DLP ou compromettre ***k*** nœuds.

- **Attaque d'épuisement de l'espace d'adressage**

Hypothèse :

- Un compteur ***RC*** est maintenu pour chaque nouveau nœud demandeur d'auto-configuration.
- Le nombre de tentatives de demande d'adresse IP est limité.

Assertion : L'attaque d'épuisement de l'espace d'adressage est infaisable.

Preuve : Un nœud demandeur malveillant désirant épuiser l'espace d'adressage, doit obtenir un nombre maximum de « certificat commun d'adresse IP et de clé publique » sans toutes fois s'enregistrer. Quand un nœud joint le réseau et sollicite une auto-configuration, en cas des pertes de message et d'échecs de nœud il se peut que son certificat ne lui parvienne pas. Pour cette raison, un certain nombre de tentatives est autorisé. A chaque demande, son compteur ***RC*** est incrémenté. Pour chaque certificat délivré, une nouvelle adresse est attribuée. S'il s'enregistre, alors il ne pourra pas obtenir un autre certificat. L'adresse IP qui sera valable dans le réseau est celle pour laquelle l'enregistrement a

été fait. Si par malveillance, le nœud veut épuiser l'espace d'adressage en multipliant les tentatives, il est stoppé grâce au compteur ***RC*** et la limite de tentatives autorisée.

- **Attaque de faux conflit d'adresses**

Puisque notre schéma est stateful, il n'exige aucun mécanisme de résolution de conflit d'adresses de type DAD, et il n'y a aucune possibilité d'attaques de faux conflit d'adresses.

- **Attaque de conflit d'adresses**

Le mécanisme d'assigner une adresse IP par une coalition plutôt que par une seule entité résout le problème de l'attaque de conflit d'adresses présente dans les approches aussi stateful et que stateless.

- **Attaque sybile**

Hypothèses :

- Chaque nœud légitime admis dans le réseau MANET détient un « certificat commun d'adresse IP et de clé publique ».
- Un nœud légitime ne peut participer activement dans le réseau que s'il s'est enregistré.

Assertion : L'attaque sybile est infaisable

Preuve : Pour mener une attaque sybile, un nœud malveillant doit s'enregistrer avec un « certificat commun d'adresse IP et de clé publique » correspondant à une identité différente. Or l'obtention d'un « certificat commun d'adresse IP et de clé publique » est subordonné à l'authentification par certificat off-line, qui est délivrée par un CA off-line supposé non accessible a l'adversaire.

- **Attaque de DoS**

L'attaque DoS de surcharge du trafic est empêchée dans le fonctionnement du protocole TCSAP par le HopLimit maximum autorisé (inférieur au seuil ***k***) dans les messages ***Config_Request***.

IV.8 Conclusion

Dans ce chapitre, nous avons proposé un nouveau protocole nommé TCSAP (Threshold Cryptography based Secure Autoconfiguration Protocol) basé sur le concept de seuil, pour fournir un service d'auto-configuration sécurisé dans les réseaux MANETs. Le protocole fournit les tâches suivantes :

- L'auto-initialisation sécurisé d'un réseau MANET autonome grâce à un sous ensemble seuil de nœuds fondateurs.

- L'auto-configuration sécurisée d'un nouveau nœud joignant le réseau. Le service est assuré par un sous ensemble seuil de nœuds honnêtes déjà configurés.
- La prise en charge de la migration des nœuds et le changement de topologie (partitionnement/fusion de réseau).

Notre solution est basée sur une authentification mutuelle, et un modèle de confiance d'Autorité de Certification et d'Auto-configuration complètement distribué avec une réplication totale de l'information d'état. Nous avons introduit un nouveau concept de certificat commun liant l'adresse IP et la clé publique à l'identité. Le déploiement de ce certificat conjointement avec un schéma de signature à seuil basée sur le problème du logarithme discret (DLP) nous a permis de contrecarrer toutes les attaques identifiées dans le chapitre 2. A travers une analyse de sécurité, nous avons démontré la sécurité et la robustesse de notre protocole sous un model d'adversaire fort. Nous avons, par ailleurs, décrit tous les mécanismes du protocole TCSAP et développé les algorithmes associés.
Dans le chapitre suivant, nous donnerons une preuve de concept par examen des performances du protocole TCSAP.

CHAPITRE 5

Evaluation des Performances du Protocole TCSAP

V.1 Introduction

Dans le chapitre précédent, nous avons fourni les spécifications du nouveau protocole d'auto-configuration proposé TCSAP et nous avons ensuite démontré sa sécurité et sa robustesse. Il est à présent nécessaire d'évaluer ses performances pour identifier les paramètres les plus pertinents, de mettre en évidence le coût de l'intégration de la sécurité et de valider en conséquence le fonctionnement du protocole. Nos investigations ont porté sur le comportement du protocole vis-à-vis du contexte de déploiement (paramètres de l'environnement) et du choix des paramètres intrinsèques du protocole.

Dans les sections suivantes, nous présentons la méthodologie d'évaluation adoptée basée sur la simulation utilisant l'outil NS2, qui sera suivie de notre implémentation sous NS2 du protocole TCSAP. Nous fournissons ensuite une description des expériences de simulation conduites. Les résultats sont enfin présentés, suivis de nos observations.

V.2 Méthodologie d'évaluation

Dans le processus de conception d'un protocole réseau, implémenter intégralement un nouveau protocole peut demander beaucoup d'effort et de temps, ainsi la phase d'évaluation est une étape

fondamentale, qui permet avant l'implémentation dans des conditions réelles, d'analyser le comportement du protocole et de l'ajuster en fonction des résultats obtenus. Trois principales méthodes d'évaluation sont aujourd'hui utilisées lors de la conception des protocoles réseaux: la méthode analytique, l'émulation et la simulation.

L'approche analytique a pour but la définition d'une façon formelle des comportements moyen et asymptotique d'un protocole. Cette méthode requiert souvent de faire abstraction des conditions réelles, entraînant à des simplifications grossières et des hypothèses trop fortes et peu fidèles à la réalité, ce qui rend souvent imprécis les résultats d'évaluation. Par ailleurs, la modélisation mathématique du comportement des protocoles distribués (cas de TCSAP) est souvent abandonnée en raison de la complexité.

L'émulation permet, quant à elle, d'éviter les contraintes d'espace lors du déploiement expérimental. Cependant, la plupart des solutions d'émulation EMPOWER [107], Emulab [108] et Seawind [109] ne permettent pas le passage à l'échelle et requiert une implémentation avancée du protocole, et éventuellement un passage de main à des techniciens, compliquant le processus de conception. Par conséquent, l'émulation ne permet ni d'obtenir des observations sous des conditions réelles, ni de simplifier le processus de conception de protocoles.

Pour le cas des réseaux MANETs contexte de cette thèse, les scenarios d'évaluation envisagés sont difficilement réalisables sur une plate-forme d'expérimentation réelle ou même une plateforme d'émulation. De ce fait, dans notre cas la simulation comme méthode de validation est incontournable. Elle nous offre la possibilité de bien maîtriser les variables de l'environnement et de parvenir, à faible coût, à des observations plus approfondies qui ne peuvent être obtenues à l'aide de modèles formels. Dans la suite de ce chapitre, nous développons donc une preuve de concept par simulation ; néanmoins, une évaluation de la probabilité de collision des adresses IP auto-assignées sera donnée analytiquement.

V.2.1 Outil de simulation

Parmi les simulateurs de réseaux existants actuellement : NS2, GloMoSim, JiST/SWANS, GTSNetS, OPNET, qui offrent tous des environnements de simulation très développés, , nous avons opté dans le cadre de cette thèse pour le simulateur NS2. Il va sans dire, que tout autre simulateur peut être utilisé pour l'évaluation.

NS2 (Network Simulator 2) est un simulateur Open Source à événements discrets, qui se compose d'une interface de programmation en tcl et d'un noyau écrit en C++. NS2, en plus des possibilités de

paramétrage des simulations (script tcl) qu'il offre, il intègre plusieurs API pour le développement de nouveaux agents, une chose qui nous a beaucoup facilité l'implémentation du protocole TCSAP.

V.2.2 Une évaluation de deux composantes de TCSAP

Pour atteindre les buts d'évaluation cités en introduction, nous nous concentrons sur une étude d'évaluation de deux composantes du protocole TCSAP. Chaque composante correspond à une opération spécifique exécutée par le protocole. Les deux opérations considérées dans nos simulations sont :

- l'auto-initialisation d'un réseau MANET
- l'auto-configuration d'un nouveau nœud joignant le réseau MANET

V.2.3 Critères d'évaluation

Pour l'évaluation des performances de notre protocole, nous utilisons les critères suivants : l'efficacité et la scalabilité.

- **L'efficacité** : qui indique qu'un nœud peut obtenir une adresse valide en peu de temps sans encourir un overhead significatif. Ceci inclut les métriques suivantes :

  - Latence d'auto-initialisation : qui représente le délai moyen d'initialisation d'un réseau MANET, il est déterminé par le temps qui s'écoule entre l'instant d'envoi du premier message ***Init_Start*** et l'instant d'obtention du dernier « certificat commun d'adresse IP et de clé publique » pour un nœud fondateur.

  - Latence d'auto-configuration: qui représente le délai moyen pour qu'un nouveau nœud joignant le réseau MANET obtienne un « certificat commun d'adresse IP et de clé publique ». Ceci inclut tous les délais possibles provoqués par les échanges de messages, par les temporisateurs et par le processus de signature à seuil. Il est déterminé par le temps qui s'écoule entre l'instant d'envoi d'un message ***Discovery_Request*** et l'instant de réception d'un message ***Config_Cert_Reply*****.** Parceque la latence va dépendre aussi du conflit et non conflit des adresses IP assignées, nous donnerons également une mesure de la probabilité de collision en fonction du paramètre M (nombre de blocs d'adresses dans la table ***FAT***).

  - Overhead de communication : cette métrique représente le nombre de paquets de contrôle (TCSAP) émis pendant chaque opération du protocole.

- **Scalabilité** : qui exige que l'exécution du protocole n'éprouve pas une dégradation significative à mesure que les paramètres suivants augmentent :
  - la taille du réseau
  - la densité du réseau
  - la charge du réseau
  - la mobilité des nœuds

  En conséquence, pour investiguer la scalabilité de TCSAP, nous évaluons les métriques décrites ci-dessus en fonction de la taille, de la densité et de la charge du réseau, ainsi que de la mobilité des nœuds.

V.2.4 Paramètres du protocole

Nous avons trois paramètres importants dans la spécification du protocole TCSAP, qui peuvent impacter ses performances : le seuil $\boldsymbol{k}$, les délais des timers et la valeur initiale prédéfini de r_k . Dans le cadre des expériences de simulation menées dans cette thèse, nous avons étudié également les performances du protocole TCSAP en fonction de ces paramètres, afin d'identifier les valeurs optimales permettant de valider le niveau de performance.

V.3 Implémentation du protocole TCSAP

Le protocole TCSAP est implémenté en C++ sous NS-2.29 [110], en créant un nouvel agent TCSAP (niveau L3) héritant de la classe **Agent** de NS-2 (figure 5.1). Le code comprend trois fichiers : *tcsap-packet.h*, *tcsap.h* et *tcsap.cc* (voir Annexe A). Dans le fichier *tcsap-packet.h*, nous avons mis toutes les structures de données et les macros liés à nos nouveaux types de paquets. Le fichier *tcsap.h* contient toutes les constantes, les déclarations des classes des Timers, des Caches et de l'agent TCSAP, et les déclarations des états des nœuds et des messages émis et reçus. Le fichier *tcsap.cc* implémente les opérations du protocole TCSAP.

Les primitives cryptographiques ont été simplement modélisées par des retards. Nous avons employé les résultats publiés par [111] pour un processeur Intel Dual Core de 1.83 gigahertz, sous Windows Vista (mode 32 bits). Nous avons considéré l'algorithme RSA-2048 et l'algorithme DSA-1024 respectivement pour la signature ordinaire et la signature à seuil.

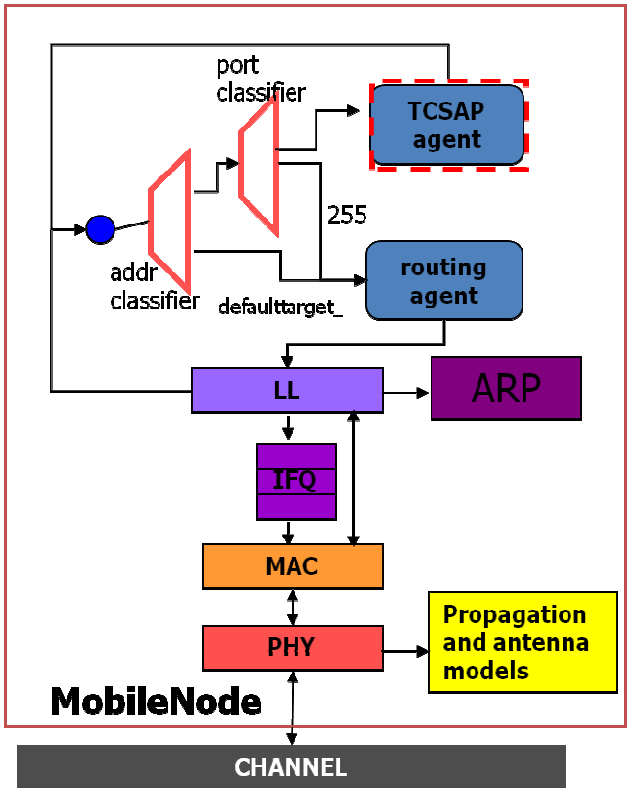


Figure 5.1 – Intégration de l'agent TCSAP dans NS2

Changement nécessaires :

1. Prise en compte des broadcasts du protocole TCSAP

   Dans le fichier « mac/ll.cc », nous modifions le code suivant :

```
void LL::recv(Packet* p, Handler* /*h*/)
{
    hdr_cmn *ch = HDR_CMN(p);
    //char *mh = (char*) HDR_MAC(p);
    //struct hdr_sr *hsr = HDR_SR(p);

    /*
     * Sanity Check
     */
    assert(initialized());

    // If direction = UP, then pass it up the stack
    // Otherwise, set direction to DOWN and pass it down the stack
    if(ch->direction() == hdr_cmn::UP) {
        //if(mac_->hdr_type(mh) == ETHERTYPE_ARP)
        if(ch->ptype_ == PT_ARP)
            arptable_->arpinput(p, this);
        else
            uptarget_ ? sendUp(p) : drop(p);
        return;
    }

    ch->direction() = hdr_cmn::DOWN;
    sendDown(p);
}
```

Par le code :

```
void LL::recv(Packet* p, Handler* /*h*/)
{
    hdr_cmn *ch = HDR_CMN(p);
    //char *mh = (char*) HDR_MAC(p);
    //struct hdr_sr *hsr = HDR_SR(p);

    /*
     * Sanity Check
     */
    assert(initialized());

    // If direction = UP, then pass it up the stack
    // Otherwise, set direction to DOWN and pass it down the stack
    if(ch->direction() == hdr_cmn::UP) {
        //if(mac_->hdr_type(mh) == ETHERTYPE_ARP)
        if(ch->ptype_ == PT_ARP)
            arptable_->arpinput(p, this);
        else {
             hdr_ip *ih = HDR_IP(p);
             if (ih->daddr() == (nsaddr_t)IP_BROADCAST)
                {
                   ih->daddr() = mac_->addr();
                   ch->next_hop() = mac_->addr();
                }
             uptarget_ ? sendUp(p) : drop(p); }
             return;
             }
    ch->direction() = hdr_cmn::DOWN;
    sendDown(p);
}
```

2. Déclaration du type de paquet TCSAP

   Dans le fichier « *common/packet.h* » , nous ajoutons le type PT_TCSAP avant le dernier type PT_NTYPE :

```
enum packet_t {
    PT_TCP,
    PT_UDP,
    .
    .
    .

    // TCSAP packet
    PT_TCSAP,

    // insert new packet types here
    PT_NTYPE // This MUST be the LAST one
};
```

Juste en-dessous, dans le même fichier il y a la définition de classe p_info. À l'intérieur du constructeur, nous fournissons un nom textuel pour notre type de paquet : name_[PT_TCSAP]= ‘‘TCSAP’’.

```
class p_info {
public:
    p_info() {
        name_[PT_TCP]= "tcp";
        name_[PT_UDP]= "udp";

    .
    .
    .


        name_[PT_TCSAP]= "TCSAP";

    }
}
```

3. Ajout du protocole TCSAP dans la librairie tcl

   Dans le fichier « *tcl/lib/ns-packet.tcl* »,

```
foreach prot {
# Common:
    Common
    Flags
    IP  # IP
.
.
.

# Mobility, Ad-Hoc Networks, Sensor Nets:
    AODV    # routing protocol for ad-hoc networks
    Diffusion   # diffusion/diffusion.cc
    IMEP    # Internet MANET Encapsulation Protocol, for ad-hoc networks
    MIP     # Mobile IP, mobile/mip-reg.cc
    Smac    # Sensor-MAC
    TORA    # routing protocol for ad-hoc networks
    TCSAP   # Threshold Cyptography based Secure Autoconfiguration Protocol
}
```

4. Valeurs par défauts des attributs liés

   Les valeurs par défaut pour des attributs liés doivent être indiquées à l'intérieur du fichier « *tcl/lib/ns-default.tcl* »

```
#...

# Defaults defined for TCSAP
Agent/TCSAP set state_ unconfigured
Agent/TCSAP set thresh_k 3
Agent/TCSAP set initial_ring 1
```

5. Makefile

Pour les besoins de compilation, nous éditons le fichier *Makefile* en ajoutant notre fichier objet *tcsap.o* à l'intérieur de la variable *OBJ_CC* :

```
OBJ_CC = \
    tools/random.o tools/rng.o tools/ranvar.o common/misc.o common/timer-handler.o \
    common/scheduler.o common/object.o common/packet.o \
    common/ip.o routing/route.o common/connector.o common/ttl.o \
    trace/trace.o trace/trace-ip.o \
    .
    .
    .
    xcp/xcpq.o xcp/xcp.o xcp/xcp-end-sys.o \
    wpan/p802_15_4csmaca.o wpan/p802_15_4fail.o \
    wpan/p802_15_4hlist.o wpan/p802_15_4mac.o \
    wpan/p802_15_4nam.o wpan/p802_15_4phy.o \
    wpan/p802_15_4sscs.o wpan/p802_15_4timer.o \
    wpan/p802_15_4trace.o wpan/p802_15_4transac.o \
    tcsap/tcsap.o \
    $(OBJ_STL)
```

V.4 Simulations

Les expériences de simulation ont été conduites en utilisant le simulateur de réseaux NS-2.29 avec les extensions de mobilité CMU et l'implémentation de TCSAP décrite ci-dessus. Le modèle de mobilité « random waypoint » a été utilisé. Le temps de simulation a été fixé à 60 secondes.

Nous avons utilisé dans nos simulations le protocole de routage AODV. Chaque point du tracé représente une valeur moyenne de cinq exécutions avec les mêmes données de simulation, mais pour différents scénarios de topologie et de mobilité aléatoirement produits. Le tableau 4.1 résume les paramètres de simulations communs à toutes nos expériences.

WirelessChannel Mac type	Mac/802_11
Phy/WirelessPhy frequency	2.4 GHz
Phy/WirelessPhy bandwidth	100 Mbps
Wireless range	250 m
Routing protocol	AODV
Mobility Model	random waypoint mobility model
Topology	variable
Threshold ***k***	3, 4, 5, 8, 10
Simulation time	60 s

Tableau 5.1- Paramètres de simulation

V.4.1 Scénarios de simulations

Conformément aux sous-sections V.2.2 et V.2.3, nous avons conduit les deux études décrites ci-après. Dans chaque étude, un ou plusieurs scénarios de simulation sont définis par des scripts tcl (voir Annexe B).

V.4.1.1 Etude 1 : Auto-initialisation d'un réseau MANET

Nous visons dans cette simulation à mesurer la latence de l'auto-initialisation d'un réseau MANET (cette latence est définie dans la sous-section V.2.3). Les impacts des paramètres suivants sont étudiés: seuil ***k*** et délai du temporisateur ***InitTimer*** (ISTR_TIMEOUT). Puisque le sous-protocole MANET_BSP s'exécute dans un voisinage à un saut, l'étude ne fait intervenir ni la taille, ni la densité ni la charge du réseau. Cela va de soit que l'évaluation de l'overhead dans ce cas est pratiquement sans intérêt.

Donc, un seul scénario (noté $S_{MANET\_BSP\_1}$) est considéré ici, dans lequel aucune mobilité des nœuds n'a été appliquée afin d'éviter que les liens entre les nœuds fondateurs ne soient rompus pendant la phase d'auto-initialisation. Les coordonnées topologiques des nœuds ont été choisies afin d'avoir chaque fois le nombre désiré de nœuds fondateurs qui correspond à la valeur du seuil ***k***=3,4,5,8,10.

V.4.1.2 Etude 2 : Auto-configuration d'un nouveau nœud joignant le réseau MANET

Dans cette étude, nous évaluons l'efficacité du sous-protocole NODE_ACP en termes de latence et d'overhead de communication, en fonction respectivement de la densité, la taille, la mobilité et la charge du réseau. L'efficacité est évaluée également en fonction des paramètres intrinsèques du protocole : seuil ***k***, délais des temporisateurs ***ConfigTimer*** et ***ConfigCertTimer***, et la valeur initiale prédéfinie de r_k . Ensuite, nous ferrons une évaluation de la probabilité de collision en fonction du taux d'arrivée des nouveaux nœuds et du paramètre M.

a) Scénario 1 ($S_{NODE\_ACP\_1}$) : Variation de la densité du réseau

Nous étudions ici l'effet de la densité du réseau sur la latence et l'overhead de communication. La taille du réseau est fixée à 300 nœuds. Aucun mouvement n'a été appliqué dans ce scénario. Nous faisons varier la topologie du réseau pour assurer la densité du réseau désirée (Tableau 5.2). Les simulations ont été effectuées également pour différentes valeurs des paramètres du protocole définis dans le Tableau 5.3.

Number of nodes	Area x=y (m)	Density (nodes/km2)
300	5477	10
300	2450	50
300	1732	100
300	1414	150
300	1224	200
300	1095	250
300	1000	300

Tableau 5.2- Paramètres de variation de la densité du réseau

Threshold ***k***	3, 5, 8, 10
r_k	1, 2, 3, ...
CREQ_TIMEOUT (ms)	250, 500, 1000
CCREQ_TIMEOUT (ms)	1000, 3000, 6000

Tableau 5.3- Paramètres du protocole TCSAP (variation de la densité)

b) Scénario 2 ($S_{NODE\_ACP\_2}$) : Variation de la taille du réseau

Dans cette simulation, nous examinons l'efficacité du protocole au passage à l'échelle. Nous considérons pour cela des réseaux de tailles variables tout en maintenant la densité à 100 nœuds/km2. Aucune mobilité ou trafic n'est introduit dans ce scénario.

Le nombre de nœuds dans le réseau avec les topologies simulées sont présentés dans le Tableau 5.5.

Number of nodes	Area x=y (m)	Density (nodes/km2)
50	707	100
100	1000	100
150	1224	100
200	1414	100
250	1581	100
300	1732	100

Tableau 5.4- Paramètres de variation de la taille du réseau

Les paramètres du protocole sont donnés dans le Tableau 5.6.

Threshold ***k***	3	5	8	10
r_k	1	1	1	1
CREQ_TIMEOUT (ms)	150	250	400	600
CCREQ_TIMEOUT (ms)	750	1000	1250	1500

Tableau 5.6- Paramètres du protocole TCSAP (variation de la taille)

c) Scénario 3 ($S_{NODE\_ACP\_3}$) : Variation de la mobilité des nœuds

Nous examinons ici l'efficacité du protocole quand la mobilité des nœuds augmente. Une topologie de réseau de 1000m sur 1000m avec 50 nœuds est simulée. Les mêmes paramètres du Tableau 5.6 sont utilisés. Nous varions la vitesse des nœuds (nodespeed) de 0 à 100 m/s suivant le modèle de mobilité « random waypoint » ; le temps de pause est fixé à 0, selon la commande suivante (cela permet de générer un fichier de mobilité pouvant être appelé par un script tcl, voir annexe B) :

Setdest –v2 –n 50 –s 1 –m (nodespeed) –M (nodespeed) –t 60 –P 1 –p 0 –x 1000 –y 1000

d) Scénario 4 ($S_{NODE\_ACP\_4}$) : Variation de la charge du réseau

Une augmentation du trafic de données est susceptible de produire plus de collisions et de perte de messages dans le réseau. Par conséquent, la charge du réseau aura un impact significatif sur l'exécution du protocole. Dans cette étude, l'effet du trafic de données sur l'efficacité en termes de latence du sous-protocole NODE_ACP est étudié. Nous utilisons un trafic de données CBR avec une augmentation du nombre de sessions. Les paramètres de simulation sont donnés dans le Tableau 5.7, et les paramètres du protocole sont ceux du Tableau 5.6.

Number of nodes	100
Topology	1000m x 1000 m
CBR Traffic PacketSize_	768 bytes
CBR Traffic Interval_	16 ms
Node speed	10 m/s
Number of sessions	5, 10, 20, 30,40 et 50

Tableau 5.7- Paramètres de simulation (variation de la charge du réseau)

e) Scénario 5 ($S_{NODE\_ACP\_5}$) : Variation du taux d'arrivée des nœuds

Dans cette partie, nous étudions la probabilité de collision d'adresses en fonction du taux d'arrivée des nœuds et le nombre de blocs disponibles. Nous admettons pour cela que le processus d'arrivée des nœuds suit une loi de Poisson :

$$p(i) = \frac{e^{-\lambda}\lambda^{i}}{i!} \qquad \textbf{(5.1)}$$

Où λ : est le taux d'arrivée des nœuds par unité de temps, la durée d'auto-configuration d'un nœud étant dans ce cas supposée égale à une seconde,

i : est le nombre de nœuds,

$p(i)$: est la probabilité d'avoir en arrivée *i* nœuds simultanément.

En utilisant le paradoxe des anniversaires, nous pouvons estimer la probabilité de collision p_{coll} pour i nœuds joignant le réseau simultanément et M blocs disponibles :

$$p_{coll} = \frac{e^{-\lambda}\lambda^{i}}{i!}\left(1 - \frac{M!}{M-i!}\frac{1}{M^{i}}\right) \quad \textbf{(5.2)}$$

L'équation (5.2) permet, pour un taux d'arrivée et une probabilité de collision donnés, de déterminer le nombre de blocs M nécessaires.

V.5 Résultats et Observations

Nous reportons dans cette section les résultats des différentes expériences de simulation conduites pour les scénarios décrits précédemment, ainsi que nos observations.

V.5.1 Résultats de l'étude 1 : Auto-initialisation

a) Délai d'auto-initialisation

Nous avons observé (figure 5.2) que le délai moyen d'initialisation augmente avec le seuil ***k***. Le protocole d'initialisation de TCSAP et le protocole distribué de partage du secret aléatoire vérifiable exécutés dans la phase d'initialisation de réseau MANET sont basés sur un nombre de messages échangés très élevé et des temps de calcul des primitives cryptographiques également élevés. Ces valeurs sont d'autant plus grandes que la valeur seuil est plus élevée. Cependant, le délai d'initialisation mesuré demeure pratiquement faible pour les valeurs seuil basses (moins de 2 secondes pour ***k***=3,4,5).

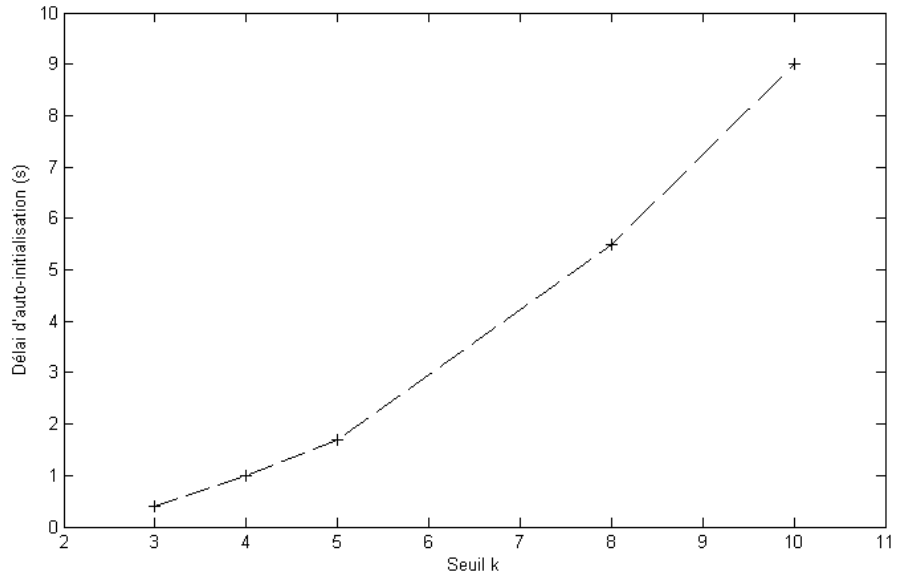


Figure 5.2 – Evolution du délai d'auto-configuration en fonction du seuil ***k***

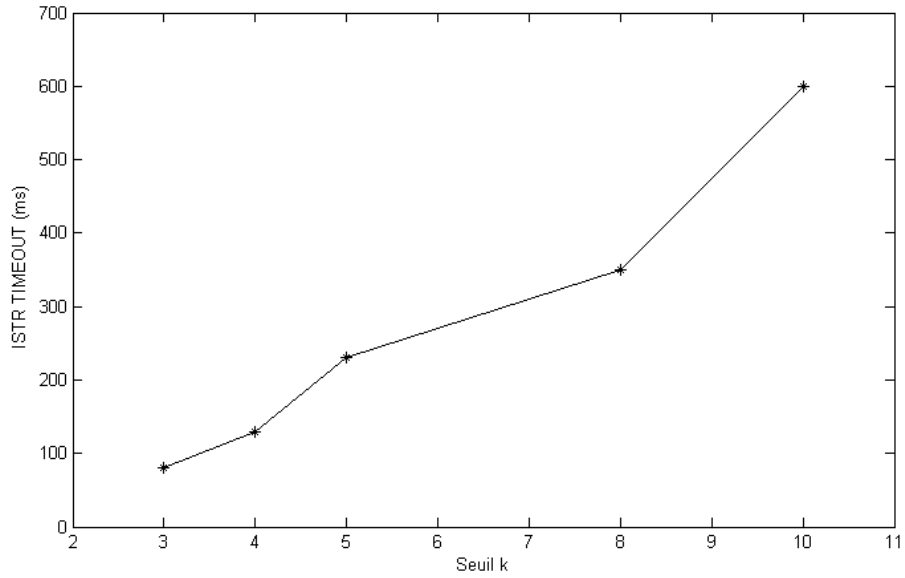


Figure 5. 3 – Mesure de la valeur optimale du paramètre ISTR_TIMEOUT en fonction du seuil ***k***

a) Valeur ISTR_TIMEOUT du temporisateur ***InitTimer***

La valeur ISTR_TIMEOUT a été fixée préalablement à la valeur par défaut, soit 500 ms pour les valeurs du seuil ***k*** = 3, 4, 5 et 8, puis à 1000 ms pour ***k***=10 ; Pour chaque expérience effectuée, nous avons relevé du fichier trace le temps s'écoulant entre l'émission d'un message ***Init_Start*** et la réception de ***k*** messages ***Init_Ack,*** les valeurs obtenues représentent les valeurs optimales pour le paramètre ISTR_TIMEOUT (figure 5.3).

V.5.2 Résultats de l'étude 2 : Auto-configuration

V.5.2.1 Impact de la densité du réseau

a) Délai d'auto-configuration

La figure 5.4 montre une augmentation du délai d'auto-configuration quand la densité du réseau est faible (en-dessous de 25 nœuds/km2), en particulier pour les valeurs seuil élevées. La valeur moyenne de la latence est moins d'une seconde. Nous avons observé un minimum à (25 nœuds/km2). La latence augmente, à nouveau à partir de ce point presque linéairement avec la densité. La latence augmente également avec le paramètre seuil. Ceci peut être justifié par le fait que le protocole dépend intiment du nombre de voisins (le degré d'un nœud). Plus la densité est importante, plus celui-ci est grand (figure 5.5), en conséquence plus de messages de contrôle des protocoles CSMA/CA et TCSAP sont échangés, ce qui retarde les messages utiles.

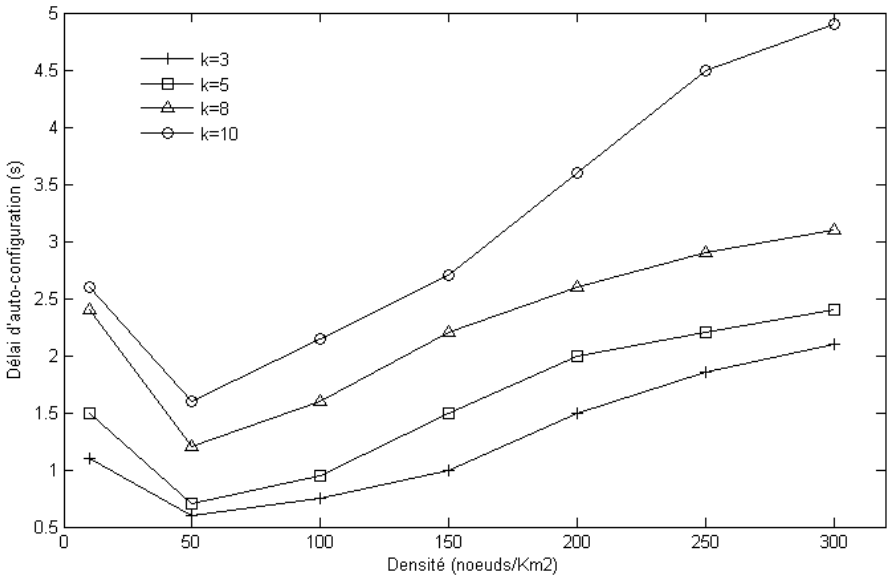


Figure 5.4 – Evolution du délai d'auto-configuration en fonction du seuil ***k***

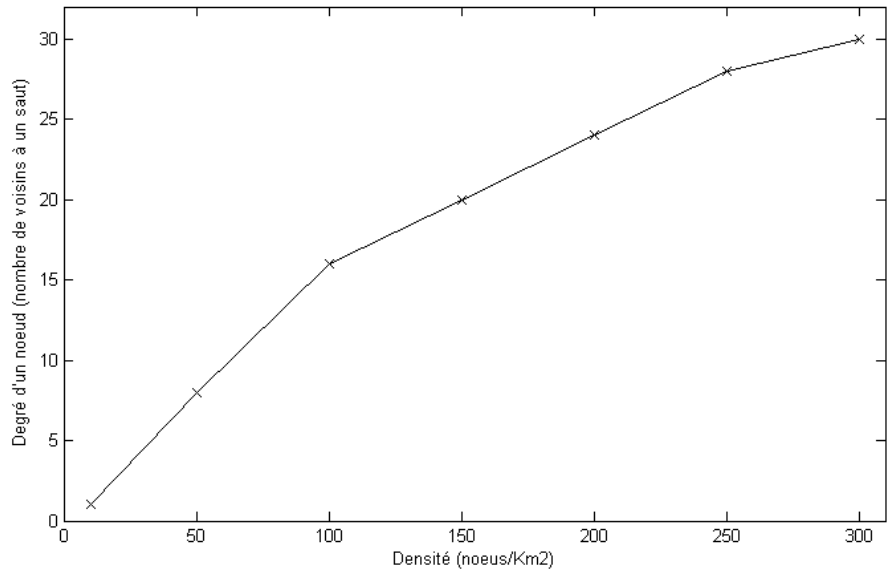


Figure 5.5 – Degré d'un nœud en fonction de la densité

b) Détermination des valeurs des paramètres CREQ_TIMEOUT, CCREQ_TIMEOUT et rk

Dans nos expériences, les valeurs des paramètres CREQ_TIMEOUT, CCREQ_TIMEOUT et rk sont testés dans un ordre croissant conformément au Tableau 5.3. Lorsque l'opération d'auto-configuration est réussie, nous prélevons du fichier trace les optimales pour ces paramètres. Ainsi, pour CREQ_TIMEOUT nous prélevons le temps s'écoulant entre l'émission par un ***Allocator*** d'un message ***Config_Request*** et la réception de ***k*** messages ***Config_Reply***, pour CCREQ_TIMEOUT nous

prélevons le temps s'écoulant entre l'émission par un ***Allocator*** du premier message unicast ***Config_Cert_Request*** et la réception du message ***Config_Cert_Reply***. Les valeurs mesurées de ces deux paramètres sont reportées respectivement sur les figures 5.6 et 5.7 pour les densités de 50 et 300 nœuds/Km^2.

La mesure de la valeur optimale pour le paramètre rk (figure 5.8) montre qu'elle est en général égale à l'unité (sauf pour le cas d'une densité de 10 nœuds/Km^2, où elle prend la valeur 3), ceci s'explique par le fait que le degré d'un nœud pour les densités considérées est dans la plupart des cas supérieur ou égal au seuil ***k*** ; au delà d'une densité de 10 nœuds/Km^2 le degré est pratiquement supérieur à 10 (figure 5.5). Donc les serveurs peuvent être trouvés avec un HopLimit égal à l'unité.

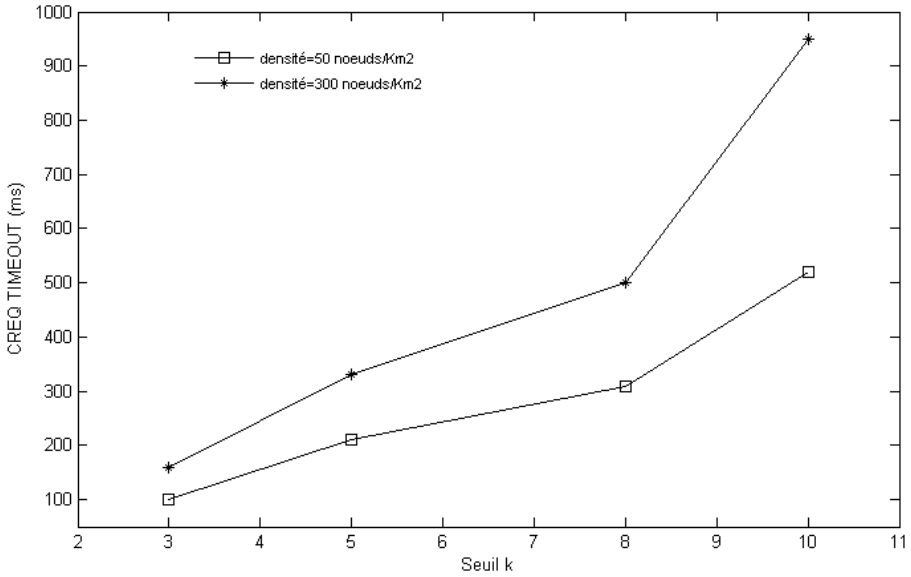


Figure 5.6 – Mesure de la valeur optimale du paramètre CREQ_TIMEOUT en fonction du seuil ***k***

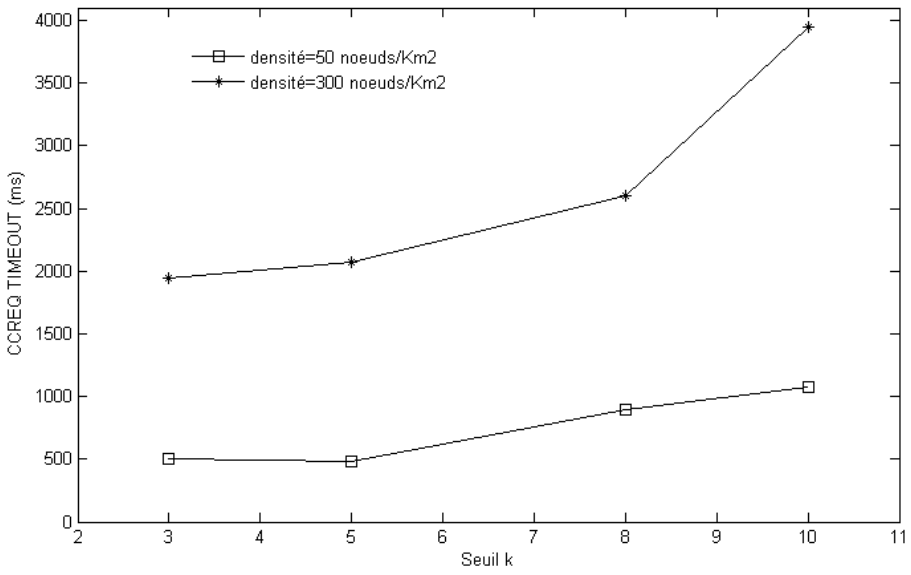


Figure 5.7 – Mesure de la valeur optimale du paramètre CCREQ_TIMEOUT en fonction du seuil ***k***

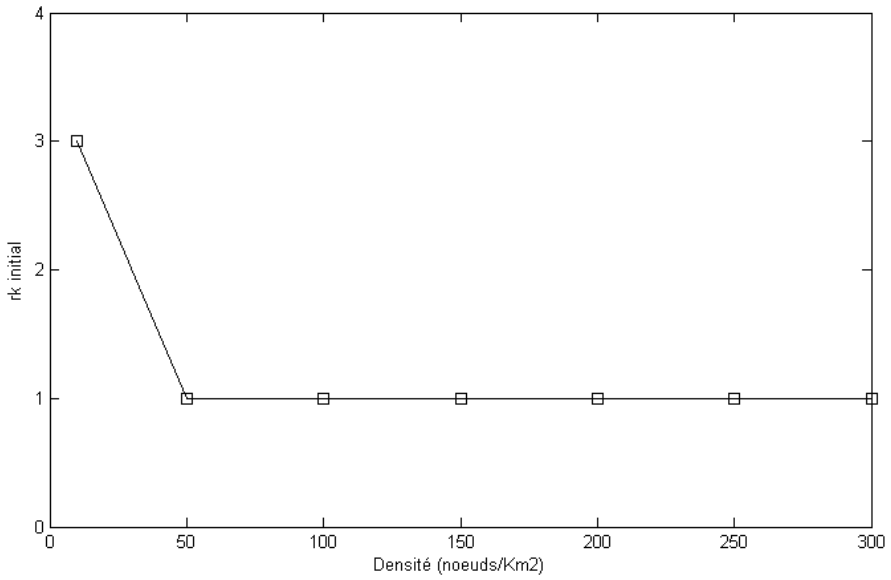


Figure 5.8 – Mesure de la valeur optimale du paramètre rk en fonction de la densité

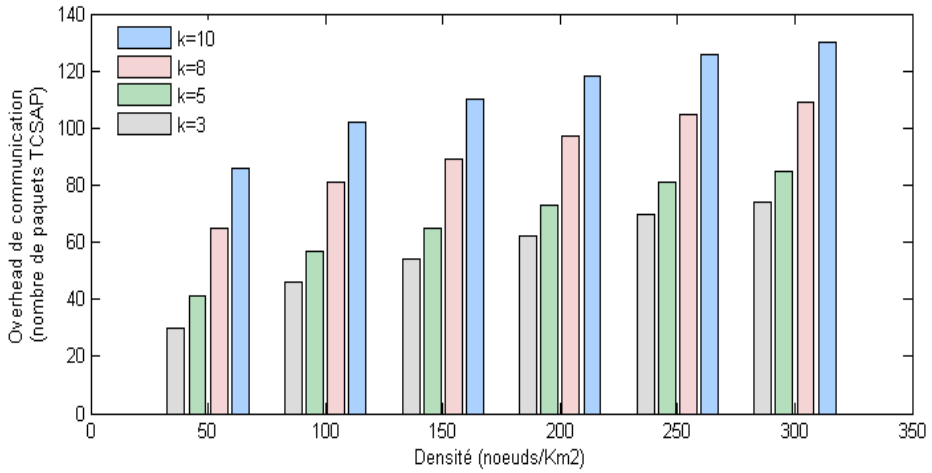


Figure 5.9 – Overhead de communication en fonction de la densité et du seuil ***k***

a) Overhead de communication

Dans les réseaux avec une densité élevée, en général il y aura plus de nœuds à proximité du nouveau nœud joignant le réseau MANET (***Requester***) et à proximité de l'***Allocator***, en conséquence ceci mène à un nombre plus élevé de messages de contrôle (paquets TCSAP). Pour cette raison, nous observons dans la figure 5.9 une augmentation de l'overhead quand la densité augmente. Notons que ceci fait augmenter également la latence d'auto-configuration. Notons que pour une densité donnée, pré-visiblement l'overhead augmente avec le seuil ***k***, ceci est dû à la nature même du protocole TCSAP qui est interactif et qui intervenir un calcul multi-parties pour la signature à seuil, conduisant à des échanges importants de messages TCSAP.

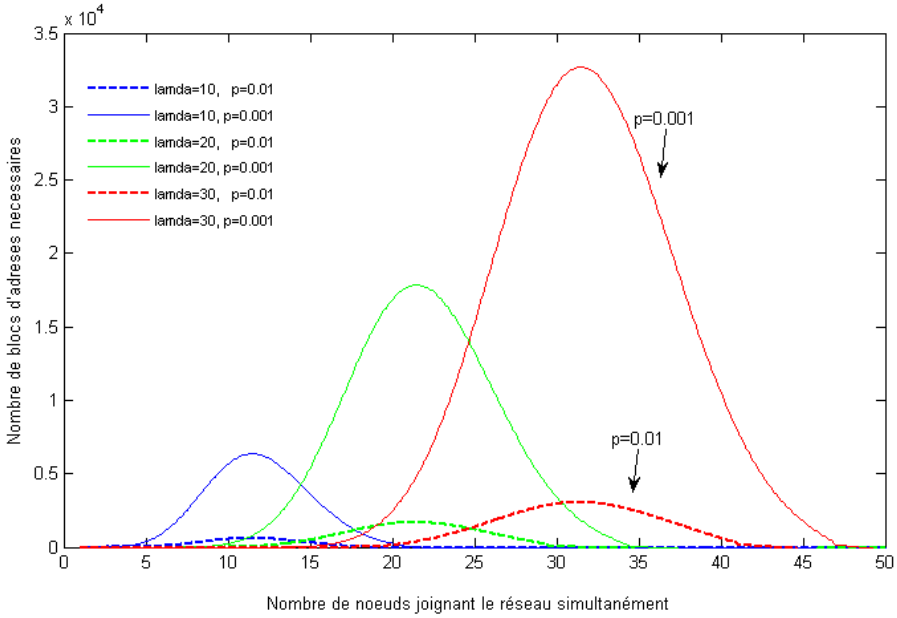


Figure 5.10 – Impact du taux d'arrivée des nœuds sur la probabilité de collision d'adresses

V.5.2.2 Impact du taux d'arrivée des nœuds

Lorsqu'une collision se produit, elle n'est détectée qu'à la phase d'enregistrement, et le processus d'auto-configuration doit être refait pour un des nœuds en conflit. Dans ce paragraphe, nous montrons qu'on peut, pour un taux d'arrivée des nœuds donné, réduire la probabilité de collision à une valeur désirée moyennant un choix adéquat du paramètre M (nombre de blocs d'adresses). En effet, compte tenu de la figure 5.10 pour un taux λ fixé, nous avons une quantification de M pour une probabilité de collision désirée.

Naturellement, M augmente lorsque p diminue. A titre d'exemple, pour λ=20 et p=10^{-3}, nous avons une valeur pour M de 18000, soit environ 150 Koctets de mémoires.

V.5.2.3 Impact de la taille du réseau

a) Délai d'enregistrement

La taille du réseau n'aucun impact sur l'auto-configuration proprement dite c'est-à-dire sur la phase d'obtention d'un « certificat commun d'adresse IP et de clé publique » par un ***Requester***, mais elle intervient dans la phase d'enregistrement, là où un message ***Config_Register*** doit être diffusé à tous les nœuds dans le réseau. La figure 5.11 donne l'évolution du délai d'enregistrement en fonction de la taille du réseau. Le délai d'enregistrement représente le temps s'écoulant entre l'émission d'un message ***Config_Register*** et sa réception par tous les nœuds du réseau MANET. Nous constatons une augmentation linéaire qui remet en cause la scalabilité du protocole.

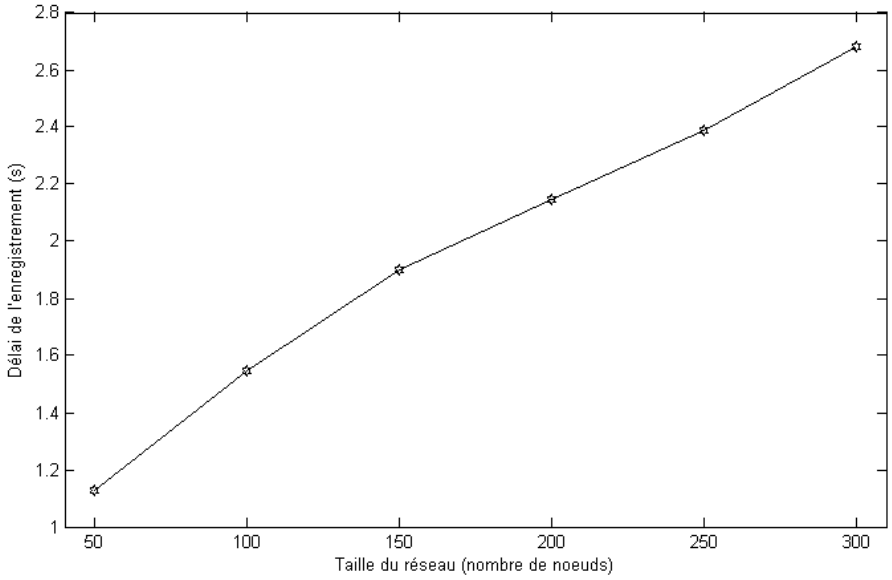


Figure 5.11 – Evolution du délai d'enregistrement en fonction de la taille du réseau

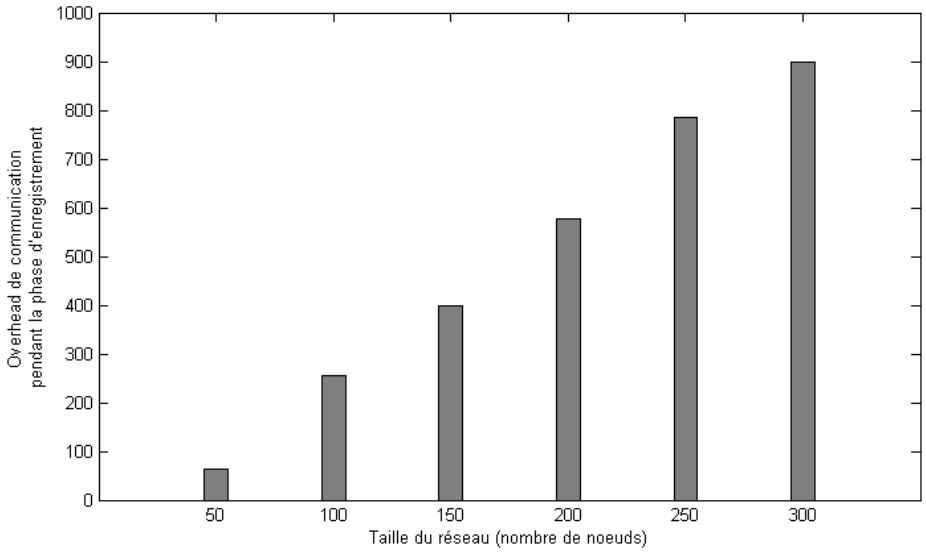


Figure 5.12 – Evolution de l'overhead de communication pendant la phase d'enregistrement en fonction de la taille du réseau

a) Overhead de communication (phase d'enregistrement)

Similairement au délai d'enregistrement, l'overhead de communication généré pendant la même phase (messages broacasts) devient très significatif avec l'augmentation de la taille du réseau (figure 5.12), ce qui limite les performances du protocole à son passage à l'échelle.

V.5.2.4 Impact de la mobilité

a) Délai d'auto-configuration

La figure 5.13 représente l'évolution du délai d'auto-configuration en fonction de la vitesse des nœuds. Rappelons qu'un modèle de mobilité aléatoire (waypoint mobility model) a été utilisé. Nous constatons que lorsque la mobilité est faible, son effet n'est pas significatif sur la latence. Ceci peut être expliqué par le fait que la latence de configuration automatique moyenne est moins de deux secondes pour les valeurs seuil considérées, et que pour des vitesses inférieures à 20m/s, le déplacement d'un nœud pour un tel délai n'excède pas 40m, et cela ne rompt pas dans la plupart des cas les liens. Lorsque la mobilité est significative (plus de 50m/s), nous constatons que le délai d'auto-configuration augmente. Ceci est dû d'une part à la rupture des liens (surtout pour les nœuds impliqués dans le service d'auto-configuration qui sont au bord du rayon de portée), et d'autre part au changement topologique qui induit des mises à jour des routes.

b) Overhead de communication

Nous constatons sur la figure 5.14 une légère augmentation de l'overhead de communication en fonction de la mobilité. Ceci s'explique par le fait que peu de messages TCSAP sont perdus à cause de la rupture des liens, et qui nécessitent une retransmission. La rupture des liens en raison de la mobilité des nœuds ne fait que retarder les émissions jusqu'à la mise à jour des tables de routage. Noter que pour une mobilité donnée, nous avons toujours une augmentation de l'overhead en fonction de la valeur du seuil.

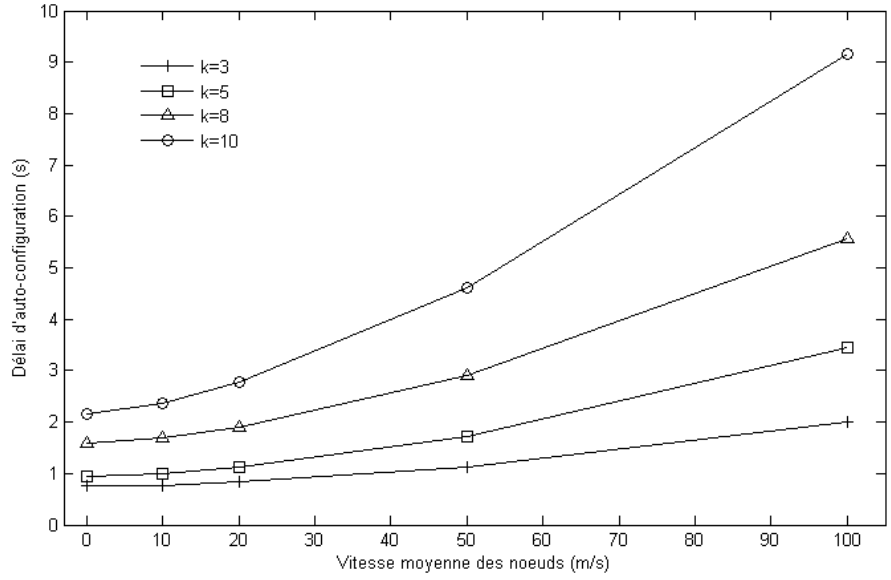


Figure 5.13 – Evolution du délai d'auto-configuration en fonction de la mobilité

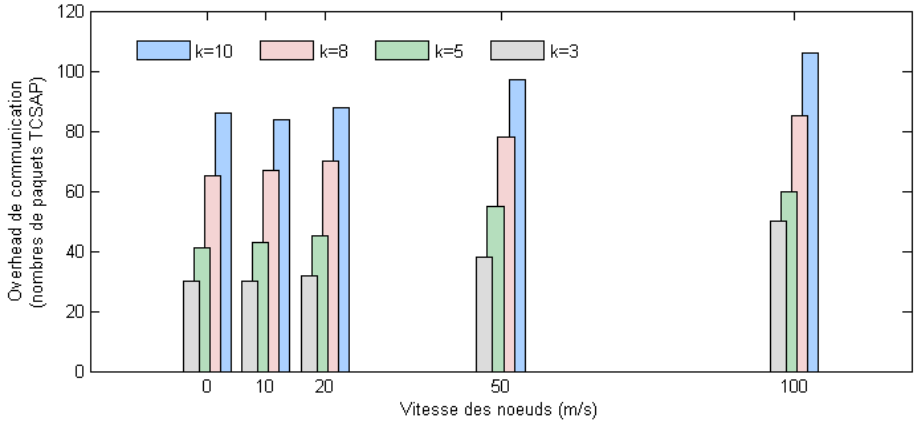


Figure 5.14 – Evolution de l'overhead de communication en fonction de la mobilité

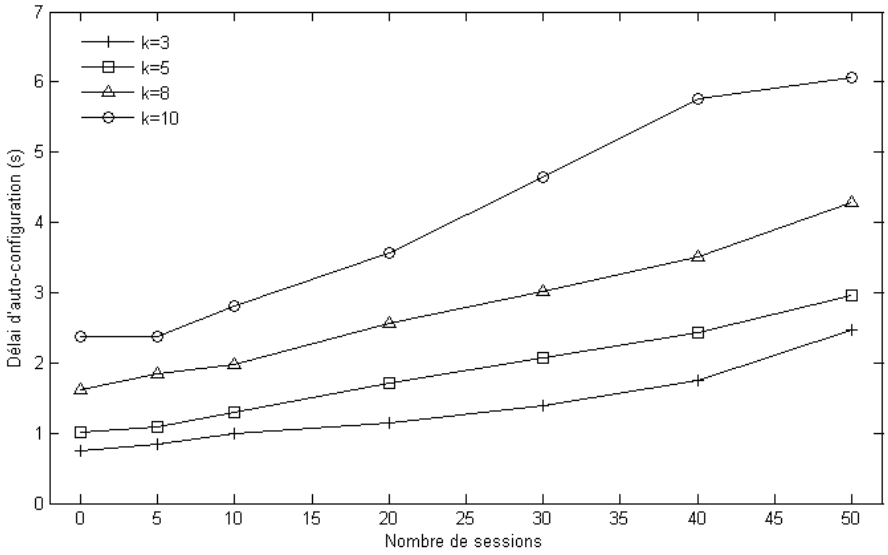
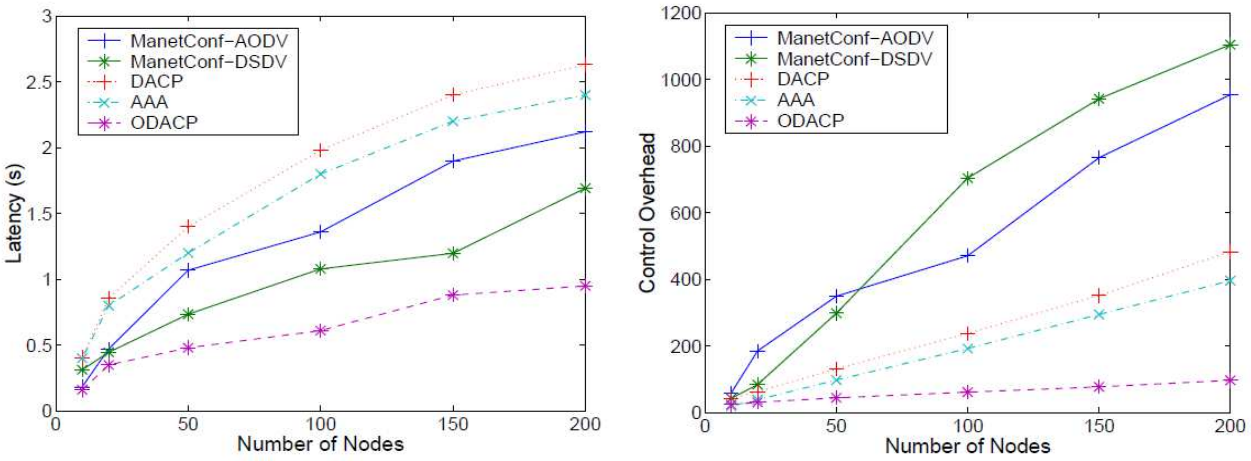


Figure 5.15 – Evolution du délai d'auto-configuration en fonction de la charge du réseau

V.5.2.5 Impact de la charge du réseau

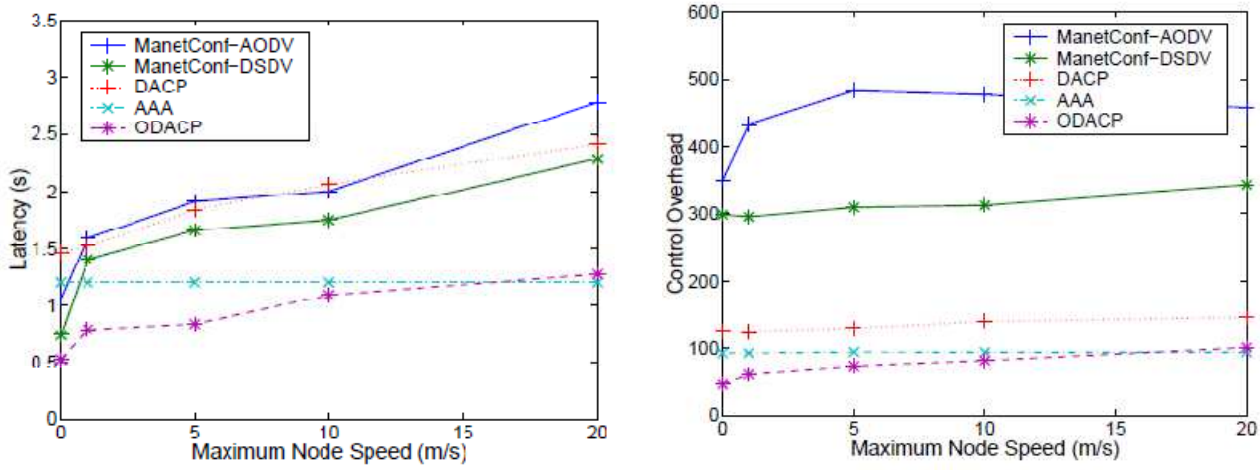
- Délai d'auto-configuration

La figure 5.15 représente l'impact du nombre de sessions ouvertes sur la latence d'auto-configuration. Il est évident que l'augmentation du trafic dans le réseau affecte considérablement l'exécution du protocole TCSAP à cause du problème des collisions du protocole aléatoire CSMA/CA, retardant ainsi les émissions des messages TCSAP, de ce fait nous observons une augmentation du délai d'auto-configuration en fonction de la charge du réseau, ce délai passe au double voir même au triple.



(a) Délai d'auto-configuration (b) Overhead de communication

Figure 5.16 – Impact de la taille du réseau (référence [51])



(a) Délai d'auto-configuration (b) Overhead de communication

Figure 5.17 – Impact de mobilité (référence [51])

V.6 Conclusion

Nous avons présenté dans ce chapitre une série d'expériences de simulation que nous avons conduites sous NS2 pour l'évaluation des performances du protocole TCSAP en termes d'efficacité et de scalabilité. Pour se faire, nous avons implémenté TCSAP sous NS2.29 et établi les scripts de simulation Tcl nécessaires.

Dans un premier temps, nous avons mesuré la latence d'auto-initialisation d'un réseau MANET par k nœuds. Nous avons constaté que cette latence augmente d'une façon polynomiale avec le seuil k. Néanmoins, des valeurs de délai pratiques sont obtenues pour des seuils faibles (moins de deux secondes pour $k \leq 5$).

Dans un deuxième temps, nous avons évalué la latence et l'overhead de contrôle engendrés par le processus d'auto-configuration d'un nouveau nœud, et cela pour différents scénarios permettant de mettre en évidence l'impact de la densité, de la mobilité, de la taille et de la charge du réseau sur ces métriques. Pour chaque cas de figure, nous avons défini les paramètres de simulation d'environnement et du protocole. Nous avons constaté que, pour un seuil ***k*** donné, la latence et l'overhead de contrôle augmentent avec la densité (plus précisément avec le degré d'un nœud), avec la mobilité et avec la charge du réseau. Il a été observé que les valeurs de ces métriques augmentent également avec le seuil ***k***. Cependant, les résultats obtenus pour les faibles valeurs de ***k*** (***k*** ≤ 5) restent acceptables (maximum trois secondes pour la latence et une centaine de paquets pour la signalisation).
Ces expériences nous ont permis de déterminer, par ailleurs, les valeurs optimales pour les temporisateurs et le paramètre rk utilisés par le protocole. Il a été constaté que ces valeurs optimales sont dépendantes du seuil ***k***.

En ce qui concerne le comportement du protocole vis-à-vis le passage à l'échelle, il a été montré que l'augmentation de la taille du réseau impacte la phase d'enregistrement seule. Nous avons noté dans ce cas un overhead très significatif (environ 900 paquets pour une taille de 300 nœuds).

Un autre aspect important est le comportement du protocole quand le taux d'arrivée des nœuds est grand. Il n'est pas souhaitable, dans ce cas, que des collisions d'adresses soient fréquentes. Nous avons pour cela fourni, dans l'hypothèse d'un taux d'arrivée de poisson, une relation formelle permettant de déterminer le nombre M de blocs d'adresses requis pour une probabilité de collision désirée.

Pour conclure, l'évaluation des performances faite dans ce chapitre pour le protocole TCSAP nous a permis d'avoir des résultats satisfaisants permettant de valider notre conception. Le protocole TCSAP avec les mécanismes de sécurité intégrés engendre une latence d'auto-configuration (quelques secondes) et une signalisation (quelques centaines de paquets si nous comptons la phase d'enregistrement et une centaine de paquets sinon) qui restent comparables aux résultats de l'étude faite dans [51] pour l'évaluation des performances de quelques protocoles d'auto-configuration non sécurisés tels que MANETconf, DACP et ODACP (figures 5.16 et 5.17). Cependant, TCSAP hérite une limitation du protocole MANETconf qui est le passage à l'échelle.

Conclusion générale

Les réseaux MANETs ont des spécificités qui n'existent pas dans d'autres réseaux traditionnels sans fil ou mobiles. Ils sont par nature sans infrastructure, donc non centralisés et plutôt fortement coopératifs de sorte que n'importe quelle tâche est réalisée grâce à la coopération d'un groupe de nœuds. Les nœuds ont généralement des ressources physiques limitées (capacité de calcul et de stockage, et réserve d'énergie). Si des dispositifs à batterie sont utilisés, la puissance de transmission et l'utilisation de processeurs affecteront directement la vie des batteries. La topologie de tels réseaux est fortement dynamique avec des liens et des routes qui changent rapidement. Les nœuds formant le réseau sont mobiles et peuvent donc joindre ou quitter le réseau spontanément avec éventuellement des crashes imprévisibles. Les liens sont sans fil avec des largeurs de bande et des capacités limitées. La largeur de bande est une ressource importante dans les réseaux sans fil. Cela est particulièrement valable pour les réseaux MANETs dont les nœuds doivent faire face à des trafics de signalisation additionnels, et doivent également agir en tant que routeurs pour les nœuds voisins. Ces liens ne sont pas fiables et sont susceptibles aux erreurs dues aux interférences et affaiblissements de transmission.

D'autres parts, les nœuds participants dans un réseau MANET opèrent souvent dans un environnement hostile, et les communications utilisent un support radio par nature ouvert, ce qui constitue une cible facile pour les violations de la sécurité qui peuvent se produire sous forme d'attaques passives ou actives. Pour conduire ces attaques, un adversaire s'appuie sur des nœuds malveillants ou compromis. Les nœuds manquent en général de protection physique adéquate et la probabilité que ces nœuds soient compromis est très haute. Si un nœud est compromis il peut compromettre le fonctionnement global du système. De ce fait, un schéma de sécurité robuste pour ce type de réseaux, en tenant compte bien sûr des particularités citées plus haut, est recommandé.

Dans cette thèse, nous avons adressé cette problématique pour deux issues encore ouvertes, que sont l'auto-initialisation d'un réseau MANET et l'auto-configuration des nœuds joignant le réseau. La première concerne le démarrage des services de sécurité (établissement de la base de confiance) et

d'auto-configuration par les premiers nœuds présents dans le réseau eux même sans l'appui d'une partie tierce. La seconde a trait au développement de mécanismes sécurisés de configuration automatique des nouveaux nœuds avec les paramètres de sécurité et de réseau nécessaires pour la participation active dans le réseau MANET. L'objectif étant de développer un protocole sécurisé et robuste qui prend en charge ces deux opérations sous l'hypothèse d'un modèle d'adversaire fort.

En effet, l'auto-configuration est un service fondamental dans les réseaux MANETs. Les nœuds ont besoin d'un certain mécanisme pour échanger des messages les uns avec les autres. Le protocole TCP/IP permet aux différents nœuds du réseau de communiquer en associant une adresse IP distincte à chaque nœud du même réseau. Dans les réseaux filaires ou sans fil avec une infrastructure, les adresses IP sont assignées correctement soit par un serveur dédié ou grâce à un mécanisme stateless.

Dans le contexte des réseaux MANETs, en raison de la topologie dynamique (mouvement constant des nœuds qui peuvent joindre ou quitter le réseau fréquemment et même simultanément), les protocoles de configuration automatique sont confrontés à divers problèmes pour garantir l'unicité des adresses IP dans un environnement topologique variable (migration, départ, arrivée, partitionnement/fusion). De ce fait, de nombreux protocoles de configuration dynamique et automatique adaptés aux spécificités MANET ont été développés. Cependant, ils demeurent tous vulnérables. En conséquence, cela peut conduire dans des environnements potentiellement hostiles à des attaques sérieuses.

Dans le cadre de cette thèse, nous avons porté une attention particulaire d'une part aux schémas d'auto-configuration existants ainsi qu'aux vulnérabilités et attaques possibles induites, et d'autres parts aux modèles de confiance développés pour les réseaux MANETs susceptibles de répondre à nos besoins. La première partie de la thèse a été consacrée donc à une étude bibliographique et une analyse de l'existant. Il s'agissait de dégager, en tout premier lieu, les contraintes et les exigences impératives de la solution envisagée, d'où l'ébauche d'un état de l'art en la matière.

L'analyse des approches proposées pour solutionner le problème de la base de confiance MANET nous a permis de constater une difficulté sérieuse pour l'aboutissement à une solution finale. En effet, parmi les exigences pour concevoir un schéma d'auto-configuration sécurisé et robuste sous l'hypothèse d'un modèle d'adversaire fort, figure l'authentification forte des participants et l'authenticité des messages. Cela ne nous a pas laissé beaucoup de choix pour le modèle de confiance, puisque la cryptographie asymétrique avec authentification par certificat s'imposait. D'autres parts, un schéma de révocation est exigé pour isoler les nœuds malveillants et révoquer leurs certificats. Parmi les solutions asymétriques existantes conçues pour les réseaux MANETs seul le modèle de PKI convient à nos besoins. Dans le contexte des réseaux MANETs qui sont par nature sans infrastructure, l'absence d'une autorité de certification centralisée représente un obstacle dans la mise en application

des protocoles asymétriques. La solution à ce problème a consisté à une émulation de PKI par distribution du rôle du CA entre les nœuds participants. Pour des raisons de disponibilité et d'efficacité nous avons opté pour le modèle plat de CA totalement distribuée. En s'appuyant sur la cryptographie à seuil, ce modèle permet de fournir un service omniprésent par distribution de la fonctionnalité du CA à chaque voisinage local. Comme les certificats doivent être signés, un schéma de signature à seuil (***k***,***n***) est nécessaire. Après avoir examiné, les différentes solutions existantes pour le partage du secret à seuil et les schémas de signature à seuil associés, nous avons construit notre solution sous l'hypothèse du problème du logarithme discret dans le groupe multiplicatif modulo un nombre premier. Ce choix ne peut être justifié que par rapport à la complexité des algorithmes mis en œuvre pour chaque approche. Pour le partage de secret à seuil, nous avons utilisé le partage basé sur l'interpolation de Lagrange de Shamir associé au protocole de Pederson pour la production aléatoire et distribuée du secret. Le schéma de signature à seuil adopté est celui basé sur la variante de DSA**,** un protocole reposant sur les calculs multipartis (MPC).

Ceci étant, nous avons développé d'autres parts une approche originale d'auto-configuration stateful inspirée du protocole MANETconf et basée sur le concept de seuil, où l'adresse IP n'est plus assignée par un seul nœud mais plutôt par une coalition de ***k*** nœuds. Le schéma permet de répondre aux exigences et objectifs tracés, il est en outre totalement distribué et plus efficace que MANETconf en raison du nouveau mécanisme d'attribution d'adresse IP sans consultation des participants du réseau.

Pour fournir la sécurité au schéma d'auto-configuration proposé, nous avons imposé à chaque adresse IP attribuée qu'elle soit signée par l'Autorité de Certification du réseau. Comme la signature du CA dans le modèle adopté est obtenue au moyen d'une signature à seuil, nous avons chargé les nœuds ayant attribué cette adresse de fournir la signature. Enfin, pour contrecarrer certaines attaques difficiles telles que l'attaque d'usurpation d'adresse et l'attaque sybile, nous avons imposé à chaque adresse IP qu'elle soit liée à l'identité du nœud, ce qui nous a amené à l'introduction d'un nouveau format de certificat : le certificat commun d'adresse IP et de clé publique. Ainsi, pour participer activement tout nœud légitime joignant le réseau MANET doit obtenir un certificat on-line de type « certificat commun d'adresse IP et de clé publique ».

La solution proposée est implémentée dans le protocole TCSAP (Threshold Cryptography based Secure Auto-configuration Protocol) dont la spécification a été décrite en détails. TCSAP se base sur un certain nombre de sous-protocoles fournissant chacun une fonctionnalité, nous citons entre-autres l'auto-initialisation du réseau, l'auto-configuration d'un nouveau nœud et la gestion du partitionnement/fusion de réseau. Contrairement à tous les schémas d'auto-configuration déjà proposés qui se basent sur un seul nœud pour initialiser le réseau ou fournir le service d'auto-configuration, notre schéma ne s'appuie sur aucune partie de confiance tierce, l'auto-initialisation dans notre protocole est réalisée par au moins ***k*** nœuds voisins, et le service de configuration automatique

est fourni par au moins ***k*** nœuds arbitraires parmi ***n***. Notre protocole fournit un service deux en un. Chaque configuration automatique d'un nouveau nœud permet de lui délivrer un « certificat commun d'adresse IP et de clé publique » signé par le CA on-line, donc un certificat valide et une adresse IP topologiquement valable et globalement unique. Notre schéma est totalement distribué de sorte que la disponibilité du service est garantie quelque soit la topologie du réseau.

Les résultats de nos expériences de simulation, conduites sous NS2, pour le protocole TCSAP proposé, sont confrontés à ceux de la littérature. Les constations montrent qu'il y a une augmentation de consommation de bande passante par le trafic de signalisation ainsi qu'une augmentation du délai d'auto-configuration. Ces augmentations sont dues essentiellement à l'intégration de la sécurité dans le schéma, et par conséquent à l'interactivité protocolaire qui dépend du paramètre de seuil ***k.*** Toutefois, le protocole reste comparativement efficace pour des valeurs de seuil plus ou moins faibles. A noter que le paramètre de seuil ***k*** doit répondre à un compromis robustesse/efficacité. Cependant, le protocole est limité, il faut le reconnaitre, quant au passage à l'échelle à cause de la procédure d'enregistrement qui nécessite des broadcasts.

Les expériences ont permis par ailleurs d'identifier les valeurs appropriées des paramètres utilisés dans notre modèle permettant de valider l'approche proposée pour un contexte de réseaux MANETs de taille moyenne. Il va sans dire que le protocole original TCSAP développé dans le travail de cette thèse permet de fournir pour un réseau MANET autonome et sous des hypothèses relaxées, un service d'auto-configuration à la fois sécurisé, robuste et efficace. Ce protocole surmonte toutes les limites de sécurité des approches proposées précédemment.

Néanmoins, plusieurs questions n'ont pas été considérées et peuvent éventuellement faire l'objet d'une continuité à ce travail, nous les résumons dans les points suivants :

- Un prolongement immédiat à ce travail serait l'évaluation des performances de TCSAP dans le cas d'un scénario de partitionnement/fusion de réseau.

- Le protocole TCSAP qui est indépendant de tout protocole de routage, utilise des mécanismes propres pour détecter les voisins. Nous sommes curieux de savoir ce que va donner en termes de performance une intégration de TCSAP dans un protocole de routage proactif.

- Dans le cadre du travail que nous avons mené, nous avons supposé un modèle de communications fiables. Or, ceci n'est pas réaliste puisque des pertes de paquets sont assez fréquentes dans un environnement sans fil. Il serait donc intéressant de remanier TCSAP en considérant un modèle de communications non fiables.

- Dans notre schéma, nous nous sommes basés sur le problème cryptographique DLP, comme perspective il serait intéressant de développer une solution sous l'hypothèse du problème de factorisation (RSA) ou du problème du logarithme discret sur courbes elliptiques (ECC-DLP) et comparer leurs performances.

- L'évaluation de la complexité de la solution proposée en termes de calcul et consommation d'énergie nécessite l'implémentation réelle des primitives cryptographiques utilisées dans le protocole. Ce volet peut donc faire l'objet d'un travail de recherche intéressent.

- Le modèle de confiance adopté dans notre solution est basé sur une PKI totalement distribuée. En relaxant un petit peu les hypothèses faites dans notre solution, on peut faire appel à un autre modèle moins complexe, en l'occurrence le modèle IBC qui a été très sollicité ces dernières années dans les recherches scientifiques portant sur la sécurité des réseaux MANETs.

- Le schéma d'auto-configuration développé dans cette thèse est basé sur le protocole MANETConf qui présente un défaut de scalabilité. Il serait intéressant comme perspective à ce travail d'appliquer les mécanismes de sécurité développés dans le cadre de TCSAP pour d'autres schémas d'auto-configuration scalables.

Bibliographie

[1] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc on-demand distance vector (AODV) routing. RFC 3561, Internet Engineering Task Force, July 2003.

[2] D. Johnson, Y. Hu, D. Maltz, RFC 4728, “The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4”, February 2007.

[3] T. Clausen and P. Jacquet. Optimized link state routing protocol. RFC 3626, Internet Engineering Task Force, September 2003.

[4] R. Ogier, F. Templin and M. Lewis. Topology dissemination Based on Reverse-Path Forwarding (TBRPF), RFC 3684, February 2004.

[5] M. Pearlman and Z. Haas. Determining the optimal configuration for the zone routing protocol. IEEE Journal on Selected Areas in Communications, 17(8) :1395–1414, August 1999.

[6] Y.-B. Ko and N. H. Vaidya, ”Location-aided routing (LAR) in mobile ad hoc networks”, in ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom98), pages 66-75, 1998.

[7] S. Nesargi and R. Prakash, “MANETconf: Configuration of Hosts in a Mobile Ad Hoc Network”, IEEE INFOCOM 2002, June 2002.

[8] Tinghui Xu and Jie Wu. Quorum Based IP Address Autoconfiguration in Mobile Ad Hoc Networks. In Proceedings of International Conference on Distributed Computing Systems Workshops (ICDCSW 2007), 2007.

[9] M. Mohsin and R. Prakash. IP Address Assignment in a Mobile Ad Hoc Network. IEEE Milcom 2002.

[10] H. Zhou, L. Ni and M. Mutka, “Prophet Address Allocation for Large Scale MANETs”, Proceedings of INFOCOM 2003, 2003.

[11] N. H. Vaidya, “Weak Duplicate Address Detection in Mobile Ad Hoc Networks,” Proc. ACM MobiHoc 2002, Lausanne, Switzerland, pp. 206–16, June 2002.

[12] K. Weniger, ”PACMAN: Passive Autoconfiguration for Mobile Ad Hoc Networks,” IEEE JSAC, Special Issue on Wireless Ad Hoc Networks, Mar. 2005.

[13] Y. Sun and E. M. Belding-Royer, ”Dynamic Address Configuration in Mobile Ad Hoc Networks,” UCSB tech. rep. 2003-11, Santa Barbara, CA, Jun. 2003.

[14] Frank Stajano and Ross Anderson. The resurrecting duckling: Security issues for ad-hoc wireless networks. In Security Protocols, 7th International Workshop Proceedings, Lecture Notes in Computer Science, 1999.

[15] N. Asokan and P. Ginzboorg, “Key Agreement in Ad Hoc Networks,” Computer Communication, vol. 23, no. 17, , pp.1627–1637, Nov. 2000.

[16] L. Eschenauer and V.D. Gligor. A key-management scheme for distributed sensor networks, 9th ACM conference on Computer and Communications Security, pp. 41-47, ACM Press, 2002.

[17] K. Hoeper and G. Gong, “Models of authentication in ad hoc networks and their related network properties”, CACR technical report 2004.

[18] H. Deng, Mukherjee and D. Agrawal. Threshold and Identity-Based Key Management and Authentication for Wireless Ad Hoc Networks. Proc. Int. Conf. Information Technology: Coding and Computing (ITCC ’04). April 2004.

[19] M. Girault. Self-certified public keys, Advances in Cryptology- EUROCRYPT '91, D.W. Davies (Ed.), LNCS 547, , pp. 490-497, Springer-Verlag, 1991.

[20] S. Capkun, L. Buttyán, and J-P Hubaux. Self-organized public-key management for mobile ad hoc networks. In Proceedings of the ACM International Workshop on Wireless Security (WiSe), 2002.

[21] K. Sanzgiri, B. Dahill, B. Neil Levine, C. Shields and E. Royer, “A Secure Routing Protocol for Ad Hoc Networks”, Proceedings of IEEE ICNP 2002, pp: 78-87, November 2002.

[22] P. Papadimitratos and Z. J. Haas, Secure Routing for Mobile Ad hoc Networks, In Proceedings of the SCS Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS 2002), San Antonio, TX, January 27-31, 2002.

[23] Y.-C. Hu, D. B. Johnson and A. Perrig, “SEAD: Secure Efficient Distance Vector Routing for Mobile Wireless Ad Hoc Networks“, Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA’02), Callicoon, New York, USA, June 20-21, 2002.

[24] Y. Hu, A. Perrig, and D.B. Jonson, “Ariadne: A Secure On-Demand Routing for Ad hoc Networks”, Proceedings of ACM MOBICOM 2002, pp:12-23, September 2002.

[25] M. G. Zapata and N. Asokan, “Securing Ad hoc Routing Protocols”, Proceedings of the 2002 ACM Workshop on Wireless Security (WiSe 2002). S. 1- 10. September 2002.

[26] Fan Hong, Liang Hong, and Cai Fu. Secure OLSR. In Proceedings of the 19th IEEE International Conference on Advanced Information Networking and Applications (AINA ’05), Tamkang University, Taiwan, March 28–30, 2005.

[27] F. Buiati, R. S. Puttini, and R. T. de Sousa Jr., “A Secure Autoconfiguration Protocol for MANET Nodes”. Lecture Notes in Computer Science, v. 3158, pp. 108-121, 2004.

[28] Ana Cavalli, Jean-Marie Orset, “Secure hosts autoconfiguration in mobile ad hoc networks”, ICDCSW.2004, pp. 809-814, 2004.

[29] Pan Wang, Douglas S. Reeves, Peng Ning, “Secure Address Autoconfiguration for Mobile Ad Hoc Networks”, MOBIQUITOUS 2005. pp. 519-522, 2005.

[30] Shenglan Hu, Chris J. Mitchell, “Improving IP address autoconfiguration security in MANETs using trust modelling”, Mobile Ad-hoc and Sensor Networks - First International Conference, MSN 2005, pp. 83-92, 2005.

[31] André Langer and Tom Kühnert, “Security issues in Address Autoconfiguration Protocols: An improved version of the Optimized Dynamic Address Configuration Protocol”. archiv.tuchemnitz.de, 2007.

[32] H. Zhou, Lionel M. Ni and Matt W. Mutka. Secure prophet address allocation for MANETs. Security and Communication Networks. Special Issue on Security in Next Generation Wireless Networks, Volume 3, Issue 1, pages 31–43, 2010.

[33] A. Abdelmalek, M. Feham and A. Taleb-Ahmed. On Recent Security Enhancements to Autoconfiguration Protocols for MANETs: Real Threats and Requirements. International Journal of Computer Science and Network Security (IJCSNS, ISSN 1738-7906), Vol.9, No.4, pp.401–407, April 2009.

[34] Abdelhafid Abdelmalek, Mohamed Feham, Zohra Slimane and Abdelmalik Taleb-Ahmed. Joint IP Address and Public Key Certificate Trust Model for Mobile Ad Hoc Networks. Book Chapter, “Ad-Hoc, Mobile and Wireless Networks”, Lectures notes in Computer Science, ISSN 0302-9743, ISBN 978-3-642-04382-6, Volume 5793, pp.385-390, Springer 2009.

[35] Z. Slimane, A. Abdelmalek, M. Feham and A. Taleb-Ahmed. Secure and Robust IPv6 Autoconfiguration Protocol for Mobile Adhoc Networks under Strong Adversary. International Journal of Computer Networks & Communications (IJCNC, ISSN 0975- 2293) , Vol.3, No.4, pp.216-235, July 2011.

[36] Abdelhafid Abdelmalek, Zohra Slimane, Mohamed Feham and Abdelmalik Taleb-Ahmed. TCSAP: A new Secure and Robust Modified MANETconf Protocol. Book Chapter, "Recent Trends in Wireless and Mobile Networks", Communications in Computer and Information Science, ISSN 1865-0929, ISBN 978-3-642-21937-5, Volume 162, Part 1, pp. 73-82., Springer 2011.

[37] Abdelhafid Abdelmalek, Zohra Slimane, Mohamed Feham and Abdelmalik Taleb-Ahmed. A Security Framework for Buddy System based MANET Address Allocation Scheme. International Journal of Computer Science Issues (IJCSI, ISSN 1694-0814), Vol.8, Issue 4, No.1, pp. 28-38, July 2011.

[38] C. Bernardos, M. Calderon and H. Moustafa, "Survey of IP address autoconfiguration mechanisms for MANETs", draft-bernardosmanet-autoconf-survey-03, April 2008.

[39] C. Perkins, J. Malinen, R. Wakikawa, E. Belding-Royer, and Y. Sun, "IP Address Autoconfiguration for Ad Hoc Networks", draft-ietfmanetautoconf-01.txt, November 2001.

[40] J. Jeong, J. Park, H. Kim, H. Jeong, and D. Kim, "Ad Hoc IP Address Autoconfiguration," IETF Internet Draft, draft-jeongadhoc-ip-addr-autoconf-06.txt. 2006.

[41] IETF Autoconf Working Group, http://datatracker.ietf.org/wg/autoconf/charter/

[42] R. Droms, "Dynamic Host Configuration Protocol", RFC 2131 IETF, March 1997.

[43] S. Deering and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", IETF RFC 2460, December 1998.

[44] A. Misra, S. Das, A. McAuley, and S. K. Das. Sun. Autoconfiguration, Registration and Mobility Management for Pervasive Computing. IEEE Personal Communications, vol. 08, Issue 04, Aug. 2001.

[45] M. Thoppian and R. Prakash, "A distributed protocol for dynamic address assignment in mobile ad hoc networks," IEEE Transactions on Mobile Computing, pp. 4-19, 2006.

[46] J.P.Sheu, S.H.Tu and L.H.Chan, "A Distributed IP Address Assignment Scheme for Ad Hoc Networks," in Proceedings of the 2005 11th International Conference on Parallel and Distributed Systems (ICPADS'05), volume 1, pages 439-445 Vol. 1, July 2005.

[47] Zero Configuration Networking, http://www.ietf.org/html.charters/zeroconfcharter.html

[48] K. Weniger, "Passive Duplicate Address Detection in Mobile Ad Hoc Networks," Proc. IEEE WCNC 2003, New Orleans, LA, Mar. 2003.

[49] K. Weniger and M. Zitterbart, "IPv6 Autoconfiguration in Large Scale Mobile Ad-Hoc Networks", European Wireless 2002, Florence, Italy, Feb. 2002.

[50] R. Puttini, L. Me, R. de Sousa, "MAE – MANET Authentication Extension for Securing Routing Protocols", 5th IEEE International Conference on Mobile and Wireless Communications Networks (MWCN2003), October 2003.

[51] Yuan Sun and Elizabeth M. Belding-Royer. A Study of Dynamic Addressing Techniques in Mobile Ad hoc Networks. Wireless Communications and Mobile Computing, vol.4, pp. 315-329, April 2004.

[52] C. Crépeau and C.R. Davis. A Certificate Revocation Scheme for Wireless Ad Hoc Networks. Proceedings of ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN '03), pp.54-61, 2003.

[53] Douglas stinson. Cryptographie, théorie et pratique. Taduction de Serge Vaudenay, Gildas Avoine et Pascal Junod. Vuibert, Paris, 2003.

[54] L. Zhou and Z.J. Haas, "Securing ad hoc networks", Network, IEEE, 1999. 13(6): pp: 24-30, 1999.

[55] Lidong Zhou, Fred B. Schneider, and Robbert Van Renesse. COCA: A secure distributed online certification authority. ACM Transactions on Computer Systems, 20(4):329–368, 2002.

[56] 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), April 2003.

[57] S. Yi and R. Kravets, "Key Management for Heterogeneous Ad Hoc Wireless Networks", Proceedings of the 10 th IEEE International Conference on Network Protocols(ICNP'02), 2002.

[58] Bing Wu, Jie Wu, Eduardo B. Fernandez and Spyros Magliveras, " Secure and Efficient Key management in Mobile Ad hoc Network", Proceedings of the 19th IEEE International Parallel and Disributed Processing Symposium (IPDPS'05), 2005.

[59] Kong, P. Zerfos, H. Luo, S. Lu, and L. Zhang. Providing Robust and Ubiquitous Security Support for MANET. In IEEE International Conference on Network Protocols, pages 251.260, Nov. 2001.

[60] Shafi Goldwasser and Mihir Bellare. Lecture Notes on Cryptography. Cambridge, Massachusetts, August 2001. http://theory.lcs.mit.edu/

[61] Bo Zhu, Feng Bao, Robert H. Deng, Mohan S. Kankanhalli, Guilin Wang, "Efficient and robust key management for large mobile ad hoc networks", Computer Networks, vol 48, no. 4, pp: 657-682, July 2005.

[62] B. Schneier. Applied Cryptography. John Wiley & Sons, Inc, 2nd edition, 1996.

[63] A. Shamir. Identity-based cryptosystems and signature schemes. In Proceedings of CRYPTO '84 on Advances in Cryptology, pages 47–53, Santa Barbara, CA, USA, 1984. Springer-Verlag New York, Inc. 1984.

[64] C. Gentry. Certificate-Based Encryption and the Certificate Revocation Problem, Advances in Cryptology- EUROCRYPT '2003, E. Biham (Ed.), LNCS 2656, pp.272-293, Springer-Verlag, 2003.

[65] A. Khalili, J. Katz, and W. Arbaugh. Toward Secure Key Distribution in Truly Ad-Hoc Networks, 2003 Symposium on Applications and the Internet Workshops (SAINT 2003), IEEE Computer Society, pp. 342-346, 2003.

[66] P. Barreto, H. Kim, B. Bynn and M. Scott. Efficient Algorithms for Pairing-Based Cryptosystems. Proc. CRYPTO 2002, pp. 354-368 .2002.

[67] Anne Marie Hegland et al., "A Survey Of Key Management In Ad Hoc Networks", IEEE Communications Surveys & Tutorials, Volume 8, Issue 3, pp:48 – 66, 2006.

[68] B. Lehane, L. Dolye and D.O. Mahony, "Ad hoc Key Management Infrastructure", IEEE Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC '05), 2005.

[69] S. Capkun, L. Buttyán, and J-P Hubaux. Small worlds in security systems: an analysis of the PGP certificate graph. In Proceedings of the 2002 Workshop on New Security Paradigms, pages 28–35, Virginia Beach, Virginia, USA, 2002.

[70] Nitesh Saxena. Decentralized Security Services. PhD Thesis, University of California, 2006.

[71] Desmedt and Y. Frankel, "Threshold cryptosystem," in Advances in Cryptology, CRYPTO'89, pp. 307–315, 1989.

[72] I. Ingemarsson and G. J. Simmons. A protocol to set up shared secret schemes without the assistance of mutually trusted party. In I. Damgard, editor, Advances in Cryptology - EUROCRYPT '90, volume 473 of Lecture Notes in Computer Science, pages 266–282. Springer-Verlag, 1991.

[73] L. Harn, "Group-oriented (t, n) threshold signature and digital multisignature," IEE Proceedings Computers and Digital Techniques, vol. 141, no. 5, pp. 307–313, 1994.

[74] P. Horster, M. Michels, and H. Peterson, "Comment: Digital signature with (t, n) shared verification based on discrete logarithms," IEE Electronics Letters, vol. 31, no. 14, p. 1137, 1995.

[75] T. S. Chen, "A specifiable verifier group-oriented threshold signature scheme based on the elliptic curve cryptosystem," Computer Standard & Interfaces, vol. 27, no. 1, pp. 33-38, 2004.

[76] X. Cheng, J. Liu, and X. Wang, "An identity-based signature and its threshold version" in 19th International Conference on Advanced Information Networking and Applications, AINA 2005, vol. 1, pp. 973–977, 2005.

[77] M. Quisquater, B. Preneel, and J. Vandewalle. On the security of the threshold scheme based on the Chinese remainder theorem. 5th Int.Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2002, volume 2274 of Lecture Notes in Computer Science, pages 199–210. Springer-Verlag, 2002.

[78] Victor Shoup. Practical threshold signatures. In Proceedings of Eurocrypt 2000, pages 207–220, 2000.

[79] M. Hwang and T. Chang. Threshold Signatures: Current Status and Key Issues. International Journal of Network Security, Vol.1, No.3, PP.123–137, Nov. 2005.

[80] A. Herzberg, S. Jarecki, H. Krawczyk, M. Yung, Proactive secret sharing or: How to cope with perpetual leakage, CRYPTO'1995, LNCS 963, pp. 339-352, Springer-Verlag 1995.

[81] Pierre Wolper. Introduction à la calculabilité, cours et exercices corrigés. Dunod, Paris, 2006.

[82] Bruno Martin. Codage, cryptologie et applications. Presses polytechniques et universitaires romandes, Lausanne, 2004.

[83] R.L. Rivest, A. Shamir and L.M. Adleman, "A method for obtaining digital signatures and public-key cryptosystems", Communications of the ACM, volume 21, pages 120-126, 1978.

[84] M.O. Rabin, "Digitalized signatures and public-key functions as intractable as factorization", MIT/LCS/TR-212, MIT Laboratory for Computer Science, 1979.

[85] H.C. Williams, "A modification of the RSA public-key encryption procedure", IEEE Transactions on Information Theory, volume 26, pages 726-729, 1980.

[86] W. Diffie and M. E. Hellman. New directions in cryptography. IEEE Transactions on Information Theory, IT-22(6):644–654, 1976.

[87] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory, 31:469–472, 1985.

[88] FIPS 186, "Digital signature standard", National Institute for Standards and Technology, http://csrc.ncsl.nist.gov/fips/

[89] C. P. Schnorr. Efficient Signature Generation by Smart Cards. Journal of Cryptology, 4(3):161-174, 1991.

[90] K. Nyberg and R. Rueppel, "Message recovery for signature schemes based on the discrete logarithm problem", Designs, Codes and Cryptography, volume 7, pages 61-81, 1996.

[91] R. Cramer, I. Damgard and J. B. Nielsen. Multiparty Computation, an Introduction. In Contemporary Cryptology, Advanced courses in Mathematics CRM Barcelona, Birkhauser, 2009.

[92] Pierre-Alain Fouque. Le partage de cl´es cryptographiques : Théorie et Pratique. Thèse de doctorat présentée à l'´Ecole Normale Supérieure, Université Paris 7, 2001.

[93] Sorin Iftene. Secret Sharing Schemes with Applications in Security Protocols. Ph.D Thesis, University of Iasi Romania, 2007.

[94] G. R. Blakley. Safeguarding cryptographic keys. In Proc. AFIPS 1979 National Computer Conference, pages 313{317. AFIPS, 1979.

[95] A. Shamir. How to Share a Secret. Communications of the ACM, 22(11):612–613, 1979.

[96] C. C. Yang, T. Y. Chang, J. W. Li, and M. S. Hwang, "Simple generalized group-oriented cryptosystems using ElGamal cryptosystem," International Journal of Informatica, vol. 14, no. 1, pp. 111–120, 2003.

[97] M. Cerecedo, T. Matsumoto and H. Imai, "Efficient and Secure Multiparty Generation of Digital Signatures Based on Discrete Logarithms", IEICE Trans. Fundamentals E76-(A4):531-545, April 1993.

[98] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust threshold DSS signatures. Information and Computation, 164(1):54–84, 2001.

[99] K. Miyazaki and K. Takaragi, "A threshold digital signature scheme for a smart card based system," IEICE Transactions on Fundamentals, vol. E84-A, no. 1, pp. 205–213, 2001.

[100] T. S.Wu, C. L. Hsu, H. Y. Lin, and P. S. Huang, "Improvement of the Miyazaki-Takaragi threshold digital signaure scheme," Information Processing Letter, vol. 88, no. 4, pp. 183–186, 2003.

[101] T. S. Chen, "A threshold signature scheme based on the elliptic curve cryptosystem," Applied Mathematics and Computation, vol. 162, no. 3, pp. 1119–1134, 2005.

[102] P. Feldman, "A practical scheme for non-interactive verifiable secret sharing," in Proc. 28th IEEE Symp. FOCS, pp. 427–437, 1987.

[103] T. P. Pedersen, "A threshold cryptosystem without a trusted party," in Advances in Cryptology, EURO- CRYPT'91, pp. 522–526, 1991.

[104] I. Damgard and K. Dupont. Efficient threshold RSA signatures with general moduli and no extra assumptions. 8th International Workshop on Theory and Practice in Public Key Cryptography, volume 3386 of Lecture Notes in Computer Science, pages 346–361. Springer, 2005.

[105] C. Park and K. Kurosawa, "New ElGamal type threshold digital signature scheme," IEICE Transactions on Fundamentals, vol. E79-A, no. 1, pp. 86–93, 1996.

[106] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280, Internet Engineering Task Force, May 2008.

[107] Pei Zheng et Lionel M. Ni. Empower: A cluster architecture supporting network emulation. IEEE Trans. Parallel Distrib. Syst., 15(7): 617–629, 2004.

[108] Emulab Emulator, http://www.emulab.net/

[109] Seawind Emulator, http://www.cs.helsinki.fi/research/iwtcp/seawind/

[110] NS-2 Network Simulator, http://www.isi.edu/nsnam/ns

[111] Speed Comparison of Popular Crypto Algorithms, http://www.cryptopp.com

[112] M. G. Gouda, "Elements of Network Protocol Design," John Wiley and Sons, 1998.

Annexe A

IMPLÉMENTATION DU PROTOCOLE TCSAP (CODE C++)

```
/* ============================================================================
   This software belonging to ABDELMALEK's thesis project: SECURITE DES RESEAUX
   SANS FIL HAUTS DEBITS: APPLICATION A L'INITIALISATION ET L'AUTO-CONFIGURATION
   DES RÈSEAUX MOBILES ADHOC ,
   implements the C++ code of protocol TCSAP:
   Threshold Cryptography based Secure Autoconfiguration Protocol.
   Copyright (c) 2011 STIC Laboratory - Tlemcen University.  All Rights Reserved.
   ============================================================================ */

/* ============================================================================
   The protocol TCSAP is implemented in three files tcsap_packet.h,
   tcsap.h and tcsap.cc
   ============================================================================ */



/* C++ code for file tcsap_packet.h  version 1.3 */


#ifndef __tcsap_packet_h__
#define __tcsap_packet_h__


//#define MAX_THRESH 10

/* ============================================================================
   Packet Formats...
   ============================================================================ */

#define TCSAPTYPE_DREQ  0x0101
#define TCSAPTYPE_DWEL  0x0102
#define TCSAPTYPE_DREP  0x0103

#define TCSAPTYPE_ISTR  0x0201
#define TCSAPTYPE_IACK  0x0202
#define TCSAPTYPE_IADV  0x0203
#define TCSAPTYPE_IOPP  0x0204

#define TCSAPTYPE_CREQ  0x0301
#define TCSAPTYPE_CREP  0x0302
#define TCSAPTYPE_CCREQ 0x0303
#define TCSAPTYPE_CCREP 0x0304
#define TCSAPTYPE_CREG  0x0307

/*
 * TCSAP Header Macros
 */

#define HDR_TCSAP(p)          ((struct hdr_tcsap*)hdr_tcsap::access(p))
#define HDR_TCSAP_DREQUEST(p) ((struct hdr_tcsap_drequest*)hdr_tcsap::access(p))
#define HDR_TCSAP_DWELCOME(p) ((struct hdr_tcsap_dwelcome*)hdr_tcsap::access(p))
#define HDR_TCSAP_DREPLAY(p)  ((struct hdr_tcsap_dreplay*)hdr_tcsap::access(p))

#define HDR_TCSAP_ISTART(p)     ((struct hdr_tcsap_istart*)hdr_tcsap::access(p))
#define HDR_TCSAP_IACK(p)       ((struct hdr_tcsap_iack*)hdr_tcsap::access(p))
#define HDR_TCSAP_IADVERT(p)  ((struct hdr_tcsap_iadvert*)hdr_tcsap::access(p))
#define HDR_TCSAP_IOPPOS(p)     ((struct hdr_tcsap_ioppos*)hdr_tcsap::access(p))
```

```
#define HDR_TCSAP_CREQUEST(p) ((struct hdr_tcsap_crequest*)hdr_tcsap::access(p))
#define HDR_TCSAP_CREPLAY(p)  ((struct hdr_tcsap_creplay*)hdr_tcsap::access(p))
#define HDR_TCSAP_CCREQUEST(p)((struct hdr_tcsap_ccrequest*)hdr_tcsap::access(p))
#define HDR_TCSAP_CCREPLAY(p) ((struct hdr_tcsap_ccreplay*)hdr_tcsap::access(p))
#define HDR_TCSAP_CREGISTER(p)((struct hdr_tcsap_cregister*)hdr_tcsap::access(p))

 /*
 * General TCSAP Header - shared by all formats
 */

 struct hdr_tcsap {
        u_int16_t        tcsap_type;


      static int offset_;
      inline static int& offset() { return offset_; }
      inline static hdr_tcsap* access(const Packet* p) {
            return (hdr_tcsap*) p->access(offset_);
      }
};

 struct hdr_tcsap_drequest {
        u_int16_t        dreq_type;
        u_int16_t        dreq_length;
        nsaddr_t         dreq_src_add;
        char             dreq_cert[1024];
        char             dreq_sign[16];

 inline u_int16_t& mdreq_type() { return dreq_type;}
 inline nsaddr_t& mdreq_src_add() { return dreq_src_add;}

 inline int size() {
  int sz = 0;
  sz = 262*sizeof(u_int32_t);
      assert (sz >= 0);
      return sz;
  }
};

struct hdr_tcsap_dwelcome {
        u_int16_t        dwel_type;
        u_int16_t        dwel_length;
        nsaddr_t         dwel_src_add;
        char             dwel_cert[1024];
        char             dwel_sign[16];

 inline u_int16_t& mdwel_type() { return dwel_type;}
 inline nsaddr_t& mdwel_src_add() { return dwel_src_add;}

 inline int size() {
  int sz = 0;
  sz = 262*sizeof(u_int32_t);
      assert (sz >= 0);
      return sz;
  }
};
```

```
struct hdr_tcsap_dreplay {
        u_int16_t        drep_type;
        u_int16_t        drep_length;
        nsaddr_t         drep_src_add;
        nsaddr_t         drep_neighbors[MAX_NEIGHB];
        char             drep_cert[1024];
        char             drep_sign[16];

 inline u_int16_t& mdrep_type() { return drep_type;}
 inline nsaddr_t& mdrep_src_add() { return drep_src_add;}

 inline int size() {
  int sz = 0;
  sz = (MAX_NEIGHB+262)*sizeof(u_int32_t);
      assert (sz >= 0);
      return sz;
  }
};

struct hdr_tcsap_istart {
        u_int16_t        istr_type;
        u_int16_t        istr_length;
        nsaddr_t         istr_src_add;
        nsaddr_t         istr_founders[MAX_NEIGHB];
        char             istr_alfabeta[16];
        char             istr_sign[16];

 inline u_int16_t& mistr_type() { return istr_type;}
 inline nsaddr_t& mistr_src_add() { return istr_src_add;}

 inline int size() {
  int sz = 0;
  sz = (MAX_NEIGHB+10)*sizeof(u_int32_t);
      assert (sz >= 0);
      return sz;
  }
};

struct hdr_tcsap_iack {
        u_int16_t        iack_type;
        u_int16_t        iack_length;
        nsaddr_t         iack_src_add;
        nsaddr_t         iack_founders[MAX_NEIGHB];
        char             iack_cert[1024];
        char             iack_alfabeta[16];
        char             iack_sign[16];

 inline u_int16_t& miack_type() { return iack_type;}
 inline nsaddr_t& miack_src_add() { return iack_src_add;}

 inline int size() {
  int sz = 0;
  sz = (MAX_NEIGHB+266)*sizeof(u_int32_t);
      assert (sz >= 0);
      return sz;
  }
```

```
};

struct hdr_tcsap_iadvert {
        u_int16_t        iadv_type;
        u_int16_t        iadv_length;
        nsaddr_t         iadv_src_add;
        char             iadv_sign[16];

 inline u_int16_t& miadv_type() { return iadv_type;}
 inline nsaddr_t& miadv_src_add() { return iadv_src_add;}

 inline int size() {
  int sz = 0;
  sz = 6*sizeof(u_int32_t);
      assert (sz >= 0);
      return sz;
  }
};

struct hdr_tcsap_ioppos {
        u_int16_t        iopp_type;
        u_int16_t        iopp_length;
        nsaddr_t         iopp_src_add;
        char             iopp_sign[16];

 inline u_int16_t& miopp_type() { return iopp_type;}
 inline nsaddr_t& miopp_src_add() { return iopp_src_add;}

 inline int size() {
  int sz = 0;
  sz = 6*sizeof(u_int32_t);
      assert (sz >= 0);
      return sz;
  }
};


 struct hdr_tcsap_crequest {
        u_int16_t        creq_type;
        u_int16_t        creq_length;
        nsaddr_t         creq_src_add;
        char             creq_cert[1024];
        char             creq_sign[16];

 inline u_int16_t& mcreq_type() { return creq_type;}
 inline nsaddr_t& mcreq_src_add() { return creq_src_add;}

 inline int size() {
  int sz = 0;
  sz = 262*sizeof(u_int32_t);
      assert (sz >= 0);
      return sz;
  }
};

 struct hdr_tcsap_creplay {
        u_int16_t        crep_type;
```

```
        u_int16_t        crep_length;
        nsaddr_t         crep_src_add;
        char             crep_ca_cert[1024];
        char             crep_cert[1024];
        char             crep_sign[16];

 inline u_int16_t& mcrep_type() { return crep_type;}
 inline nsaddr_t& mcrep_src_add() { return crep_src_add;}

 inline int size() {
  int sz = 0;
  sz = 518*sizeof(u_int32_t);
      assert (sz >= 0);
      return sz;
  }
};

struct hdr_tcsap_ccrequest {
        u_int16_t        ccreq_type;
        u_int16_t        ccreq_length;
        nsaddr_t         ccreq_src_add;
        nsaddr_t         ccreq_coallition[MAX_THRESH];
        u_int32_t        ccreq_lowest_hid[2];
        char             ccreq_sign[16];

 inline u_int16_t& mccreq_type() { return ccreq_type;}
 inline nsaddr_t& mccreq_src_add() { return ccreq_src_add;}

 inline int size() {
  int sz = 0;
  sz = (MAX_THRESH + 8)*sizeof(u_int32_t);
      assert (sz >= 0);
      return sz;
  }
};

 struct hdr_tcsap_ccreplay {
        u_int16_t        ccrep_type;
        u_int16_t        ccrep_length;
        nsaddr_t         ccrep_src_add;
        char             ccrep_thresh_cert[1024];
        char             ccrep_sign[16];

 inline u_int16_t& mccrep_type() { return ccrep_type;}
 inline nsaddr_t& mccrep_src_add() { return ccrep_src_add;}

 inline int size() {
  int sz = 0;
  sz = 262*sizeof(u_int32_t);
      assert (sz >= 0);
      return sz;
  }
};

struct hdr_tcsap_cregister {
        u_int16_t        creg_type;
        u_int16_t        creg_length;
```

```
        nsaddr_t         creg_src_add;
        char             creg_thresh_cert[1024];
        char             creg_sign[16];

 inline u_int16_t& mcreg_type() { return creg_type;}
 inline nsaddr_t& mcreg_src_add() { return creg_src_add;}

 inline int size() {
  int sz = 0;
  sz = 262*sizeof(u_int32_t);
      assert (sz >= 0);
      return sz;
  }
};

union hdr_all_tcsap {
      hdr_tcsap              thead;

      hdr_tcsap_drequest     drq;
      hdr_tcsap_dwelcome     dwl;
      hdr_tcsap_dreplay      drp;

      hdr_tcsap_istart       ist;
      hdr_tcsap_iack         iak;
      hdr_tcsap_iadvert      iad;
      hdr_tcsap_ioppos       iop;

      hdr_tcsap_crequest     crq;
      hdr_tcsap_creplay      crp;
      hdr_tcsap_ccrequest    ccrq;
      hdr_tcsap_ccreplay     ccrp;
      hdr_tcsap_cregister    crg;
};


#endif /* __tcsap_packet_h__ */
```

```
/*
=============================================================================
   This software belonging to ABDELMALEK's thesis project: SECURITE DES RESEAUX
   SANS FIL HAUTS DEBITS: APPLICATION A L'INITIALISATION ET L'AUTO-CONFIGURATION
   DES RÈSEAUX MOBILES ADHOC ,
   implements the C++ code of protocol TCSAP:
   Threshold Cryptography based Secure Autoconfiguration Protocol.
   Copyright (c) 2011 STIC Laboratory - Tlemcen University. All Rights Reserved.
   =============================================================================
*/

/*
=============================================================================
   The protocol TCSAP is implemented in three files tcsap_packet.h,
   tcsap.h and tcsap.cc
   =============================================================================
*/



/* C++ code for file tcsap.h  version 1.3  */


#ifndef __tcsap_h__
#define __tcsap_h__


#include "packet.h"
#include "agent.h"
#include "timer-handler.h"
#include <random.h>

#include <assert.h>
#include <lib/bsd-list.h>

#define DREQ_TIMEOUT       1.0
#define DNEIB_TIMEOUT      0.5
#define DWEL_TIMEOUT       0.5
#define ISTR_TIMEOUT       2.0
#define CREQ_TIMEOUT       1.0
#define CCREQ_TIMEOUT      5.0
#define CREG_TIMEOUT       5.0

#define ORDI_SIGN_DELAY  0.00605
#define ORDI_VERIF_DELAY  0.00016
#define THRES_SIGN_DELAY  0.01062
#define THRES_VERIF_DELAY  0.0128

#define MAX_THRESH 20
#define MAX_NEIGHB 20
#define JITTER (Random::uniform()*0.5)

class TCSAP;

/* Timers */

class DiscoveryTimer : public TimerHandler {
```

```
public:
        DiscoveryTimer(TCSAP* a) : TimerHandler() {
        a_ = a ;
        }
        //void    handle(Event*);
        void expire(Event* e);
protected:
        TCSAP*    a_;

};


class NeighborsTimer : public TimerHandler {

public:
        NeighborsTimer(TCSAP* a) : TimerHandler() {
        a_ = a ;
        }
        //void    handle(Event*);
        void expire(Event* e);
protected:
        TCSAP*    a_;

};

class WelcomeTimer : public TimerHandler {
public:
        WelcomeTimer(TCSAP* a) : TimerHandler() {
        a_ = a ;
        }
        //void    handle(Event*);
        void expire(Event* e);
protected:
        TCSAP*    a_;

};

class InitTimer : public TimerHandler {
public:
        InitTimer(TCSAP* a) : TimerHandler() {
        a_ = a ;
        }
        //void    handle(Event*);
        void expire(Event* e);
protected:
        TCSAP*    a_;

};

class ConfigTimer : public TimerHandler {
public:
        ConfigTimer(TCSAP* a) : TimerHandler() {
        a_ = a ;
        }
        //void    handle(Event*);
        void expire(Event* e);
protected:
```

```
        TCSAP*    a_;

};


class ConfigCertTimer : public TimerHandler {
public:
        ConfigCertTimer(TCSAP* a) : TimerHandler() {
        a_ = a ;
        }
        //void    handle(Event*);
        void expire(Event* e);
protected:
        TCSAP*    a_;

};


class RegistrationTimer : public TimerHandler {
public:
        RegistrationTimer(TCSAP* a) : TimerHandler() {
        a_ = a ;
        }
        //void    handle(Event*);
        void expire(Event* e);
protected:
        TCSAP*    a_;

};

class CQueryTimer : public TimerHandler {
public:
        CQueryTimer(TCSAP* a) : TimerHandler() {
        a_ = a ;
        }
        //void    handle(Event*);
        void expire(Event* e);
protected:
        TCSAP*    a_;

};


/* TCSAP Neighbors cache Entry */

class TCSAP_Neighbors {
      friend class TCSAP;
      friend class NeighborsTimer;
   public:
          TCSAP_Neighbors(u_int32_t a) {n_add = a ;}
   protected:
          LIST_ENTRY(TCSAP_Neighbors) t_neighbors;
          nsaddr_t n_add;
          nsaddr_t nbor[MAX_NEIGHB];
};
LIST_HEAD(tcsap_ncache, TCSAP_Neighbors);
```

```
/* TCSAP WelcomeReply cache Entry */

class TCSAP_Wel {
      friend class TCSAP;
   public:
          TCSAP_Wel(u_int32_t a) {w_add = a ;}
   protected:
          LIST_ENTRY(TCSAP_Wel) t_wel;
          nsaddr_t w_add;
};
LIST_HEAD(tcsap_wcache, TCSAP_Wel);



/* TCSAP Servers cache Entry */

class TCSAP_Server {
      friend class TCSAP;
   public:
          TCSAP_Server(u_int32_t a) {s_add = a ;}
   protected:
          LIST_ENTRY(TCSAP_Server) t_server;
          nsaddr_t s_add;
};
LIST_HEAD(tcsap_scache, TCSAP_Server);


class TCSAP_Client {
      friend class TCSAP;
   public:
          TCSAP_Client(u_int32_t a) {c_add = a ;}
   protected:
          LIST_ENTRY(TCSAP_Client) t_client;
          nsaddr_t c_add;
          double   expire;
          bool     fwd;
};
LIST_HEAD(tcsap_ccache, TCSAP_Client);


class TCSAP_Vcert {
      friend class TCSAP;
   public:
          TCSAP_Vcert(u_int32_t a) {v_add = a ;}
   protected:
          LIST_ENTRY(TCSAP_Vcert) t_vcert;
          nsaddr_t v_add;
};
LIST_HEAD(tcsap_vcache, TCSAP_Vcert);

/* Agent TCSAP */

enum NodeState {unconfigured, configured, inprogress};
```

```
class TCSAP : public Agent {

/*make some friends first */

        friend class TCSAP_Neighbors;
        friend class TCSAP_Wel;
        friend class TCSAP_Server;
        friend class TCSAP_Client;
        friend class TCSAP_Vcert;

        friend class DiscoveryTimer;
        friend class NeighborsTimer;
        friend class WelcomeTimer;
        friend class InitTimer;
        friend class ConfigTimer;
        friend class ConfigCertTimer;
        friend class RegistrationTimer;
        friend class CQueryTimer;

/* Private members */

    nsaddr_t        node_adr;
    int             state_;
    int             kthresh;
    int             initial_ring;
    int             curent_ring;
    bool            manetInitiated;

 public:
        TCSAP(nsaddr_t);

        void            recv(Packet *p, Handler *);


 protected:

        int         command(int, const char *const *);
        void        start_Discovery();

        void        n_insert(nsaddr_t id, nsaddr_t neib[MAX_NEIGHB]);
        TCSAP_Neighbors*    n_lookup(nsaddr_t id);
        void        n_delete(nsaddr_t id);
        void        n_purge(void);
        int         n_size(void);

        void        w_insert(nsaddr_t id);
        TCSAP_Wel* w_lookup(nsaddr_t id);
        void        w_delete(nsaddr_t id);
        void        w_purge(void);
        int         w_size(void);

        void        s_insert(nsaddr_t id);
        TCSAP_Server* s_lookup(nsaddr_t id);
        void        s_delete(nsaddr_t id);
        void        s_purge(void);
        int         s_size(void);
```

```
        void        c_insert(nsaddr_t id);
        TCSAP_Client* c_lookup(nsaddr_t id);
        void        c_delete(nsaddr_t id);
        void        c_purge(void);

        void        v_insert(nsaddr_t id);
        TCSAP_Vcert* v_lookup(nsaddr_t id);
        void        v_delete(nsaddr_t id);
        void        v_purge(void);
        int         v_size(void);


        DiscoveryTimer          d_timer;
        NeighborsTimer          n_timer;
        WelcomeTimer            w_timer;
        InitTimer               i_timer;
        ConfigTimer             c_timer;
        ConfigCertTimer         cc_timer;
        RegistrationTimer       r_timer;
        CQueryTimer             qc_timer;

        tcsap_ncache            nhead;
        tcsap_wcache            whead;
        tcsap_scache            shead;
        tcsap_ccache            chead;
        tcsap_vcache            vhead;

        inline nsaddr_t&        node_address() { return node_adr; }
        inline int&             k_threshold() { return kthresh; }
        inline int&             initial_rg() { return initial_ring; }
        int       state() { return state_; }


/*
         * Packet RX Routines
         */

        void recvTCSAP(Packet* p);
        void recvDRequest(Packet* p);
        void recvDWelcome(Packet* p);
        void recvDReplay(Packet* p);
        void recvIStart(Packet* p);
        void recvIAck(Packet* p);
        //void recvIAdvert(Packet* p);
        //void recvIOppos(Packet* p);
        void recvCRequest(Packet* p);
        void recvCReplay(Packet* p);
        void recvCCRequest(Packet* p);
        void recvCCReplay(Packet* p);
        void recvCRegister(Packet* p);
        void forwardCQuery(Packet* p);

/*
         * Packet TX Routines
         */

        void sendDRequest();
```

```
        void sendDWelcome(nsaddr_t dweldest);
        void sendDReplay(nsaddr_t drepneighbors[MAX_NEIGHB]);
        void sendIStart(nsaddr_t istrfounders[MAX_NEIGHB]);
        void sendIAck(nsaddr_t iackfounders[MAX_NEIGHB]);
        void sendIAdvert();
        void sendIOppos();
        void sendCRequest();
        void sendCReplay(nsaddr_t crepdest);
        void sendCCRequest(nsaddr_t ccreqdest,
                           nsaddr_t ccreqcoallition[MAX_THRESH]);
        void sendCCReplay(nsaddr_t ccrepdest);
        void sendCRegister();
};


#endif  /* __tcsap_h__ */
```

```
/*
=============================================================================
   This software belonging to ABDELMALEK's thesis project: SECURITE DES RESEAUX
   SANS FIL HAUTS DEBITS: APPLICATION A L'INITIALISATION ET L'AUTO-CONFIGURATION
   DES RÈSEAUX MOBILES ADHOC ,
   implements the C++ code of protocol TCSAP:
   Threshold Cryptography based Secure Autoconfiguration Protocol.
   Copyright (c) 2011 STIC Laboratory - Tlemcen University. All Rights Reserved.
   =============================================================================
*/

/*
=============================================================================
   The protocol TCSAP is implemented in three files tcsap_packet.h,
   tcsap.h and tcsap.cc
   =============================================================================
*/



/* C++ code for file tcsap.cc  version  1.3 */


#include <packet.h>
#include <object.h>
#include <agent.h>
#include <trace.h>
#include <scheduler.h>
#include <ip.h>
#include "timer-handler.h"

#include <sys/types.h>
#include <config.h>
#include <assert.h>
#include <lib/bsd-list.h>

#include <tcsap/tcsap.h>
#include <tcsap/tcsap_packet.h>

#include <cmu-trace.h>


#define CURRENT_TIME    Scheduler::instance().clock()
#define DEBUG




/*
  TCL Hooks
*/


int hdr_tcsap::offset_;
static class TCSAPHeaderClass : public PacketHeaderClass {
public:
        TCSAPHeaderClass() : PacketHeaderClass("PacketHeader/TCSAP",
                                               sizeof(hdr_all_tcsap)) {
```

```
        bind_offset(&hdr_tcsap::offset_);
      }
} class_TCSAP_hdr;


static class TCSAPclass : public TclClass {
public:
        TCSAPclass() : TclClass("Agent/TCSAP") {}
        TclObject* create(int argc, const char*const* argv) {
          assert(argc == 5);
        return (new TCSAP((nsaddr_t) Address::instance().str2addr(argv[4])));
        }
} class_TCSAP;


double Init_starttime;
double Autoconfig_starttime;

/*
  Timers
*/



void
DiscoveryTimer::expire(Event *e)
{
  a_->state_=0;
  a_->n_purge();
  a_->w_purge();
  a_->start_Discovery();
}

void
NeighborsTimer::expire(Event *e)
{

  int p=0;
  int myneigh=1;
  nsaddr_t x[MAX_NEIGHB][MAX_NEIGHB];
  nsaddr_t y[MAX_NEIGHB];
  nsaddr_t z[MAX_NEIGHB];
  nsaddr_t fnd[a_->kthresh];

  // Initialization
  for (int i=0;i<MAX_NEIGHB;i++){
        for (int j=0;j<MAX_NEIGHB;j++){x[i][j]=-1;}
      }
  for (int i=0;i<MAX_NEIGHB;i++) {y[i]=-1; z[i]=-1;}

  // Transfert from Neighbors Cache

  int m=0;
  TCSAP_Neighbors *nb = a_->nhead.lh_first;
  for (;nb;nb=nb->t_neighbors.le_next) {

      x[m][0]= nb->n_add;
      for (int n=1;n<MAX_NEIGHB;n++)
```

```
        {x[m][n]= nb->nbor[n];}
        m++;
    }

    // Find goog neighbors ---> y[]---> z[]

    for (int l=0;l<a_->n_size();l++){
              int k=0;
         for (int i=0;i<a_->n_size();i++){

              for (int j=0;j<MAX_NEIGHB;j++){
                  if (x[l][0]==x[i][j])
                  {y[l]=x[l][0];
                  k++;}
              }
         }
         if (k>= a_->kthresh-1){z[p]=y[l];
         fprintf(stderr," z[%d]= %d \n",p,z[p]);
         p++;
         }
      }

    //select Founders

    for (int i=0;i<a_->n_size();i++){
           if (z[i]!=-1) myneigh++;
           fprintf(stderr," myneigh = %d \n",myneigh);
           }

    if (myneigh >= a_->kthresh) {
                    fnd[0]=a_->node_adr;
                    for (int i=1;i<a_->kthresh;i++)
                    {fnd[i]=z[i-1];
                    fprintf(stderr," fnd[%d]= %d\n",i,fnd[i]);
                    }

    a_->state_=2;
    a_->sendIStart(fnd);
    a_->d_timer.cancel();
    a_->i_timer.resched(ISTR_TIMEOUT);
    Init_starttime = Scheduler::instance().clock();

}
}

void
WelcomeTimer::expire(Event *e)
{

    /*int wcount;
    wcount= a_->w_size();
    fprintf(stderr," WCOUNT = %d \n",wcount);
    a_->w_purge();
    if (wcount >= (a_->kthresh)/2){a_->initial_ring=1;}
    else {a_->initial_ring=int((a_->kthresh/wcount));}
    fprintf(stderr," INITIAL RING = %d \n",a_->initial_ring);*/

    a_->curent_ring = a_->initial_ring;
```

```
    a_->sendCRequest();
    a_->d_timer.cancel();
    a_->c_timer.resched(CREQ_TIMEOUT);
    a_->state_=2;


}

void
InitTimer::expire(Event *e)
{
 a_->state_=0;
 a_->n_purge();
 a_->start_Discovery();

}


void
ConfigTimer::expire(Event *e)
{

  if (a_->curent_ring==a_->kthresh) {
        a_->state_=0;
        a_->start_Discovery();
  }
  else {

       a_->s_purge();
       a_->curent_ring +=1;
       a_->sendCRequest();
       a_->c_timer.resched(CREQ_TIMEOUT);
  }
}

void
ConfigCertTimer::expire(Event *e)
{
  a_->sendCRequest();
  a_->c_timer.resched(CREQ_TIMEOUT);
}

void
RegistrationTimer::expire(Event *e)
{
  a_->state_=1;
}

void
CQueryTimer::expire(Event *e)
{
  a_->c_purge();
}
```

```
/*
   Constructor
*/

TCSAP::TCSAP(nsaddr_t node_id) : Agent(PT_TCSAP),
                  d_timer(this), n_timer(this), w_timer(this),i_timer(this),
              c_timer(this), cc_timer(this),
                  r_timer(this), qc_timer(this){

  bind("state_", &state_);
  bind("kthresh", &kthresh);
  bind("initial_ring", &initial_ring);
  node_adr = node_id;

  LIST_INIT(&nhead);
  LIST_INIT(&whead);
  LIST_INIT(&shead);
  LIST_INIT(&chead);
  LIST_INIT(&vhead);

}


int
TCSAP::command(int argc, const char*const* argv) {

  if (argc == 2)
    {

      if (strcasecmp(argv[1], "start") == 0)
      {
      Autoconfig_starttime = Scheduler::instance().clock();
      start_Discovery();
        return TCL_OK;
      }
    }

  return (Agent::command (argc, argv));
}

void
TCSAP::start_Discovery() {
     if (state()== 0) {
        manetInitiated = false;
        sendDRequest();
        d_timer.resched(DREQ_TIMEOUT);

        //Autoconfig_starttime = Scheduler::instance().clock();

        }
}



/* Implementing Neighbors cache */

void
TCSAP::n_insert(nsaddr_t id, nsaddr_t neib[MAX_NEIGHB]) {
```

```
TCSAP_Neighbors *nb = new TCSAP_Neighbors(id);
assert(nb);

// definir les autres elements de la cache nb->x =
for (int i=0;i<MAX_NEIGHB;i++) {
nb->nbor[i]=neib[i];
}

LIST_INSERT_HEAD(&nhead, nb, t_neighbors);
}

TCSAP_Neighbors*
TCSAP::n_lookup(nsaddr_t id) {
TCSAP_Neighbors *nb = nhead.lh_first;
for (;nb;nb=nb->t_neighbors.le_next) {
    if (nb->n_add == id)  break;
    }
return nb;
}

void
TCSAP::n_delete(nsaddr_t id) {
TCSAP_Neighbors *nb = nhead.lh_first;
for (;nb;nb=nb->t_neighbors.le_next) {
       if (nb->n_add == id) {
          LIST_REMOVE(nb, t_neighbors);
          delete nb;
          break;
          }
}
}

void
TCSAP::n_purge() {
TCSAP_Neighbors *nb = nhead.lh_first;
TCSAP_Neighbors *nbn;
for (;nb;nb=nbn) {
    nbn = nb->t_neighbors.le_next;
    n_delete(nb->n_add);
}
}

int
TCSAP::n_size() {
TCSAP_Neighbors *nb = nhead.lh_first;
int i=0;
for (;nb;nb=nb->t_neighbors.le_next) {
    i +=1;
}
return i;
}


/* Implementing Servers cache */

void
TCSAP::w_insert(nsaddr_t id) {
TCSAP_Wel * wl= new TCSAP_Wel(id);
```

```
assert(wl);

// definir les autres elements de la cache sv->x =

LIST_INSERT_HEAD(&whead, wl, t_wel);
}

TCSAP_Wel*
TCSAP::w_lookup(nsaddr_t id) {
TCSAP_Wel *wl = whead.lh_first;
for (;wl;wl=wl->t_wel.le_next) {
    if (wl->w_add == id)  break;
    }
return wl;
}

void
TCSAP::w_delete(nsaddr_t id) {
TCSAP_Wel *wl = whead.lh_first;
for (;wl;wl=wl->t_wel.le_next) {
       if (wl->w_add == id) {
          LIST_REMOVE(wl, t_wel);
          delete wl;
          break;
          }
}
}

void
TCSAP::w_purge() {
TCSAP_Wel *wl = whead.lh_first;
TCSAP_Wel *wln;
for (;wl;wl=wln) {
    wln = wl->t_wel.le_next;
    w_delete(wl->w_add);
}
}

int
TCSAP::w_size() {
TCSAP_Wel *wl = whead.lh_first;
int i=0;
for (;wl;wl=wl->t_wel.le_next) {
    i +=1;
}
return i;
}

/* Implementing Servers cache */

void
TCSAP::s_insert(nsaddr_t id) {
TCSAP_Server *sv = new TCSAP_Server(id);
assert(sv);

// definir les autres elements de la cache sv->x =

LIST_INSERT_HEAD(&shead, sv, t_server);
```

```
}

TCSAP_Server*
TCSAP::s_lookup(nsaddr_t id) {
TCSAP_Server *sv = shead.lh_first;
for (;sv;sv=sv->t_server.le_next) {
    if (sv->s_add == id)  break;
    }
return sv;
}

void
TCSAP::s_delete(nsaddr_t id) {
TCSAP_Server *sv = shead.lh_first;
for (;sv;sv=sv->t_server.le_next) {
       if (sv->s_add == id) {
          LIST_REMOVE(sv, t_server);
          delete sv;
          break;
          }
}
}

void
TCSAP::s_purge() {
TCSAP_Server *sv = shead.lh_first;
TCSAP_Server *svn;
for (;sv;sv=svn) {
    svn = sv->t_server.le_next;
    s_delete(sv->s_add);
}
}

int
TCSAP::s_size() {
TCSAP_Server *sv = shead.lh_first;
int i=0;
for (;sv;sv=sv->t_server.le_next) {
    i +=1;
}
return i;
}

/* Implementing Client cache */

void
TCSAP::c_insert(nsaddr_t id) {
TCSAP_Client *cl = new TCSAP_Client(id);
assert(cl);
cl->expire = CURRENT_TIME + CREQ_TIMEOUT;
cl->fwd = true;

LIST_INSERT_HEAD(&chead, cl, t_client);
}

TCSAP_Client*
TCSAP::c_lookup(nsaddr_t id) {
TCSAP_Client *cl = chead.lh_first;
```

```
for (;cl;cl=cl->t_client.le_next) {
    if (cl->c_add == id)  break;
    }
return cl;
}

void
TCSAP::c_delete(nsaddr_t id) {
TCSAP_Client *cl = chead.lh_first;
for (;cl;cl=cl->t_client.le_next) {
       if (cl->c_add == id) {
          LIST_REMOVE(cl, t_client);
          delete cl;
          break;
          }
}
}

void
TCSAP::c_purge() {
TCSAP_Client *cl = chead.lh_first;
TCSAP_Client *cln;
double now = CURRENT_TIME;
for (;cl;cl=cln) {
    cln = cl->t_client.le_next;
   if(cl->expire <= now)
    c_delete(cl->c_add);
}
}


/* Implementing Vcert cache */

void
TCSAP::v_insert(nsaddr_t id) {
TCSAP_Vcert *vc = new TCSAP_Vcert(id);
assert(vc);

// definir les autres elements de la cache sv->x =

LIST_INSERT_HEAD(&vhead, vc, t_vcert);
}

TCSAP_Vcert*
TCSAP::v_lookup(nsaddr_t id) {
TCSAP_Vcert *vc = vhead.lh_first;
for (;vc;vc=vc->t_vcert.le_next) {
    if (vc->v_add == id)  break;
    }
return vc;
}

void
TCSAP::v_delete(nsaddr_t id) {
TCSAP_Vcert *vc = vhead.lh_first;
for (;vc;vc=vc->t_vcert.le_next) {
       if (vc->v_add == id) {
          LIST_REMOVE(vc, t_vcert);
```

```
          delete vc;
          break;
          }
}
}

void
TCSAP::v_purge() {
TCSAP_Vcert *vc = vhead.lh_first;
TCSAP_Vcert *vcn;
for (;vc;vc=vcn) {
    vcn = vc->t_vcert.le_next;
    v_delete(vc->v_add);
}
}

int
TCSAP::v_size() {
TCSAP_Vcert *vc = vhead.lh_first;
int i=0;
for (;vc;vc=vc->t_vcert.le_next) {
    i +=1;
}
return i;
}


/*************************************************************************/

void
TCSAP::recv(Packet *p, Handler*) {
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih = HDR_IP(p);

 if(ih->saddr()==node_adr) {

        drop(p, DROP_RTR_ROUTE_LOOP);
       return;
 }

 if(ch->ptype() == PT_TCSAP) {
   recvTCSAP(p);

   return;
 }
}



void
TCSAP::recvTCSAP(Packet *p) {
 struct hdr_tcsap *th = HDR_TCSAP(p);

 switch(th->tcsap_type) {

 case TCSAPTYPE_DREQ:
   recvDRequest(p);
   break;
```

```
 case TCSAPTYPE_DWEL:
   recvDWelcome(p);
   break;

 case TCSAPTYPE_DREP:
   recvDReplay(p);
   break;

 case TCSAPTYPE_ISTR:
   recvIStart(p);
   break;

 case TCSAPTYPE_IACK:
   recvIAck(p);
   break;

 case TCSAPTYPE_IADV:
   recvIAdvert(p);
   break;

 case TCSAPTYPE_IOPP:
   recvIOppos(p);
   break;

 case TCSAPTYPE_CREQ:
   recvCRequest(p);
   break;

 case TCSAPTYPE_CREP:
   recvCReplay(p);
   break;

 case TCSAPTYPE_CCREQ:
   recvCCRequest(p);
   break;

 case TCSAPTYPE_CCREP:
   recvCCReplay(p);
   break;

 case TCSAPTYPE_CREG:
   recvCRegister(p);
   break;

 default:
   fprintf(stderr, "Invalid TCSAP type (%x)\n", th->tcsap_type);
   exit(1);
 }

}


void
TCSAP::recvDRequest(Packet* p) {
//struct hdr_ip *ih = HDR_IP(p);
struct hdr_tcsap_drequest     *dq = HDR_TCSAP_DREQUEST(p);
```

```
  /*
   * Drop if I'm the source
   */

  if(dq->dreq_src_add == node_adr) {
#ifdef DEBUG
    fprintf(stderr, "got my own DISCOVERY REQUEST\n");
#endif // DEBUG
    Packet::free(p);
    return;
  }

  if (state()== 1) {

        sendDWelcome(dq->dreq_src_add);
        Packet::free(p);
        }


  if ((state()== 0) && (manetInitiated==false)) {

          nsaddr_t neib[MAX_THRESH];
              int i=0;
          TCSAP_Neighbors *nb = nhead.lh_first;
          for (;nb;nb=nb->t_neighbors.le_next) {

              neib[i]= nb->n_add;

              i++;
          }

          sendDReplay(neib);
          Packet::free(p);
          }


  else
  Packet::free(p);
}

void
TCSAP::recvDReplay(Packet* p) {
//struct hdr_ip *ih = HDR_IP(p);
struct hdr_tcsap_dreplay     *dp = HDR_TCSAP_DREPLAY(p);

/*
   * Drop if I'm the source
   */

  if(dp->drep_src_add == node_adr) {
#ifdef DEBUG
    fprintf(stderr, "got my own Discovery REPLAY\n");
#endif
    Packet::free(p);
    return;
  }
```

```
  if ((manetInitiated==false)&&(state()== 0)) {
     if (n_size()==0) {n_timer.resched(DNEIB_TIMEOUT);}
     fprintf(stderr, "node %d RECEIVE DISCOVERY REPLAY from %d at %.8f \n",
     node_adr, dp->drep_src_add, Scheduler::instance().clock());

     TCSAP_Neighbors* nb;
     nb = n_lookup(dp->drep_src_add);
     if (n_timer.status()== 1) {

        if (nb == 0) {
        nsaddr_t    neibrs[MAX_NEIGHB];
        for (int i=0;i<MAX_NEIGHB;i++)   { neibrs[i]= dp->drep_neighbors[i];  }
        n_insert(dp->drep_src_add, neibrs);
        }

     }

     Packet::free(p);

  }
  else
  Packet::free(p);
}

void
TCSAP::recvDWelcome(Packet* p) {
//struct hdr_ip *ih = HDR_IP(p);
struct hdr_tcsap_dwelcome     *dw = HDR_TCSAP_DWELCOME(p);



/*
   * Drop if I'm the source
   */

  if(dw->dwel_src_add == node_adr) {
#ifdef DEBUG
    fprintf(stderr, "got my own DISCOVERY WELCOME\n");
#endif // DEBUG
    Packet::free(p);
    return;
  }

  if ((state()== 0) &&(d_timer.status()== 1)) {

        if (w_size()==0) {
        w_timer.resched(DWEL_TIMEOUT);}

        if (manetInitiated==false) {
        manetInitiated=true;
        if (n_timer.status()== 1) {n_timer.cancel();}
        }

        TCSAP_Wel* wel ;
        wel= w_lookup(dw->dwel_src_add);

        if ((w_timer.status()== 1)) {
            if (wel == 0) {
```

```
                        w_insert(dw->dwel_src_add);

     fprintf(stderr, "Receive DISCOVERY WELCOME from %d at %.8f , w_size = %d\n"
     ,dw->dwel_src_add, Scheduler::instance().clock(), w_size());
         if (w_size()>= kthresh){w_timer.resched(0.0);}
                             }
        }

        Packet::free(p);

        }
  else
  Packet::free(p);
}


void
TCSAP::recvIStart(Packet* p) {
//struct hdr_ip *ih = HDR_IP(p);
struct hdr_tcsap_istart     *is = HDR_TCSAP_ISTART(p);



/*
   * Drop if I'm the source
   */

  if(is->istr_src_add == node_adr) {
#ifdef DEBUG
    fprintf(stderr, "got my own InitStart\n");
#endif // DEBUG
    Packet::free(p);
    return;
  }



      // if((manetInitiated==false)&&(is->istr_src_add < node_adr)) {
      // if (i_timer.status()== 1) {i_timer.cancel();(state_= 0);}
       //}

  if (/*(state()== 0)&&*/(manetInitiated==false)/*&&(i_timer.status()!= 1)*/) {
        fprintf(stderr, "node %d receive InitStart from %d\n",
        node_adr, is->istr_src_add);

        for (int j=0;j<kthresh;j++) {
            if(is->istr_founders[j] == node_adr) {
              if (n_timer.status()== 1) n_timer.cancel();
              if (d_timer.status()== 1) d_timer.cancel();
               nsaddr_t founders[MAX_NEIGHB];
               for (int i=0;i<kthresh;i++) {founders[i]= is->istr_founders[i];}
               sendIAck(founders);
               state_= 2;
               i_timer.resched(ISTR_TIMEOUT);
            }
        }
```

```
  Packet::free(p);

  }
  else
  Packet::free(p);
}

void
TCSAP::recvIAck(Packet* p) {
//struct hdr_ip *ih = HDR_IP(p);
struct hdr_tcsap_iack     *ia = HDR_TCSAP_IACK(p);



/*
   * Drop if I'm the source
   */

  if(ia->iack_src_add == node_adr) {
#ifdef DEBUG
    fprintf(stderr, "got my own InitAck\n");
#endif // DEBUG
    Packet::free(p);
    return;
  }
fprintf(stderr, "node %d receive InitAck from %d\n",node_adr, ia->iack_src_add);
  if (manetInitiated==false) {

        for (int j=0;j<kthresh;j++) {
            if(ia->iack_founders[j] == node_adr) {

               state_=1;
               manetInitiated=true;
               if (i_timer.status()== 1) i_timer.cancel();
               double Init_endtime = Scheduler::instance().clock();
fprintf(stderr, "Node %d MANET INITIATED sucessfully !!!! Init delay = %.8f\n",
                node_adr, Init_endtime-Init_starttime);
fprintf(stderr, "MANET INITIATED sucessfully with  %d  nodes\n", kthresh);
           for (int j=0;j<kthresh;j++)
           fprintf(stderr, " Founding node %d  \n", ia->iack_founders[j]);
            }
        }


  Packet::free(p);

}
  else
  Packet::free(p);
}


void
TCSAP::recvCRequest(Packet* p) {
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih = HDR_IP(p);
struct hdr_tcsap_crequest     *cq = HDR_TCSAP_CREQUEST(p);
```

```
if(ch->direction() == hdr_cmn::DOWN) {
    Packet::free(p);
    return;
  }


/*
   * Drop if I'm the source
   */



  if(cq->creq_src_add == node_adr) {
#ifdef DEBUG
    fprintf(stderr, "got my own CONFIGURE REQUEST\n");
#endif
    Packet::free(p);
    return;
  }

double now = CURRENT_TIME;
TCSAP_Client* cli ;
cli = c_lookup(cq->creq_src_add);

  if (state()== 1) {
              if (cli == 0) {

                      c_insert(cq->creq_src_add);
                      qc_timer.resched(CREQ_TIMEOUT);
                      sendCReplay(cq->creq_src_add);
                      fprintf(stderr, "TTL %x\n",ih->ttl_);
                      /*&& ((cli->fwd) != 1))*/
                      if (((ih->ttl_)-1)>0)  {
                              forwardCQuery(p);
                              fprintf(stderr, "TEST OK\n");
                              }
                      else
                          Packet::free(p);
                      }

              else if (cli->expire <= now) {
                       cli->expire = CURRENT_TIME + CREQ_TIMEOUT;
                       qc_timer.resched(CREQ_TIMEOUT);
                       sendCReplay(cq->creq_src_add);
                       if ((((ih->ttl_)-1)>0) /*&& ((cli->fwd) != 1)*/) {
                              forwardCQuery(p);
                              }
                       else
                           Packet::free(p);
                       }
                   else
                       Packet::free(p);
  }
  else
  Packet::free(p);
}

void
```

```
TCSAP::recvCReplay(Packet* p) {
//struct hdr_ip *ih = HDR_IP(p);
struct hdr_tcsap_creplay     *cp = HDR_TCSAP_CREPLAY(p);

/*
   * Drop if I'm the source
   */

  if(cp->crep_src_add == node_adr) {
#ifdef DEBUG
    fprintf(stderr, "got my own CONFIGURE REPLAY\n");
#endif
    Packet::free(p);
    return;
  }

  fprintf(stderr, "node %d RECEIVE CONFIGURE REPLAY from %d\n",
  node_adr, cp->crep_src_add);

TCSAP_Server* svr ;
svr = s_lookup(cp->crep_src_add);
if ((state()== 2) && (c_timer.status()== 1)) {


        /* check timer */

        if (svr == 0) {
        s_insert(cp->crep_src_add);


        }

        if (s_size()==kthresh) {
                              fprintf(stderr, "OK Threshold CREPLAY\n");
           c_timer.cancel();
           int i=0;
           nsaddr_t coal[MAX_THRESH];

           TCSAP_Server *sv = shead.lh_first;
           for (;sv;sv=sv->t_server.le_next) {

               coal[i]= sv->s_add;

               i++;
           }
           sv = shead.lh_first;
           for (;sv;sv=sv->t_server.le_next) {
           fprintf(stderr, "CCRequest DEST %x\n", sv->s_add);
           sendCCRequest(sv->s_add, coal);
           }

           cc_timer.resched(CCREQ_TIMEOUT);
        }

        Packet::free(p);

  }
  else
```

```
  Packet::free(p);
}

void
TCSAP::recvCCRequest(Packet* p) {
//struct hdr_ip *ih = HDR_IP(p);
struct hdr_tcsap_ccrequest     *ccq = HDR_TCSAP_CCREQUEST(p);




/*
   * Drop if I'm the source
   */

  if(ccq->ccreq_src_add == node_adr) {
#ifdef DEBUG
    fprintf(stderr, "got my own CONFIGURE CERTIFICATE REQUEST\n");
#endif
    Packet::free(p);
    return;
  }
  if (state_== 1) {


        sendCCReplay(ccq->ccreq_src_add);

        Packet::free(p);
        }
  else
  Packet::free(p);
}


void
TCSAP::recvCCReplay(Packet* p) {
//struct hdr_ip *ih = HDR_IP(p);
struct hdr_tcsap_ccreplay     *ccp = HDR_TCSAP_CCREPLAY(p);

/*
   * Drop if I'm the source
   */

  if(ccp->ccrep_src_add == node_adr) {
#ifdef DEBUG
    fprintf(stderr, "got my own CONFIGURE CERTIFICATE REPLAY\n");
#endif
    Packet::free(p);
    return;
  }



TCSAP_Vcert *vct ;
```

```
vct = v_lookup(ccp->ccrep_src_add);

  /* check state && timer */

 if ((state()== 2) /*&& (cc_timer.tatus()!= 1)*/) {

        if (vct == 0) {
        v_insert(ccp->ccrep_src_add);
        }

        if (v_size()==kthresh) {
           cc_timer.cancel();

           double Autoconfig_endtime = Scheduler::instance().clock();
        fprintf(stderr, "Autoconfiguration sucessful !!!! Config delay = %.2f\n"
        , Autoconfig_endtime-Autoconfig_starttime);
           sendCRegister();
           r_timer.resched(CREG_TIMEOUT);
        }

        Packet::free(p);
  }
  else
  Packet::free(p);
}



void
TCSAP::recvCRegister(Packet* p) { }


void
TCSAP::forwardCQuery(Packet* p) {
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih  = HDR_IP(p);

//ih->ttl_ -= 1;
ih->daddr()=IP_BROADCAST;
ch->direction() = hdr_cmn::DOWN;
Scheduler::instance().schedule(target_, p, JITTER);

}

void
TCSAP::sendDRequest() {
Packet *p          = allocpkt();
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih  = HDR_IP(p);
struct hdr_tcsap_drequest     *sdq = HDR_TCSAP_DREQUEST(p);

#ifdef DEBUG
fprintf(stderr, "sending Discovery Request from %d at %.8f\n",
node_adr, Scheduler::instance().clock());
#endif // DEBUG

 sdq->dreq_type = TCSAPTYPE_DREQ;
```

```
 sdq->dreq_length = 0x00AA;
 sdq->dreq_src_add = node_adr;
 strcpy(sdq->dreq_cert,cert_);
 strcpy(sdq->dreq_sign,sign_);

 // ch->uid() = 0;
 ch->ptype() = PT_TCSAP;
 ch->size() = IP_HDR_LEN + sdq->size();
 //ch->iface() = -2;
 //ch->error() = 0;
 //ch->addr_type() = NS_AF_NONE;
 //ch->prev_hop_ = node_adr;
 //ch->direction() = hdr_cmn::DOWN;

 ih->saddr() = node_adr;
 ih->daddr() = IP_BROADCAST;
 ih->sport() = 0; //RT_PORT;
 ih->dport() = 0; //RT_PORT; //ih->sport();
 ih->ttl_ = 1;

 Scheduler::instance().schedule(target_, p, ORDI_SIGN_DELAY);
}

void
TCSAP::sendDReplay(nsaddr_t drepneighbors[MAX_NEIGHB]) {
Packet *p          = allocpkt();
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih  = HDR_IP(p);
struct hdr_tcsap_dreplay     *sdp = HDR_TCSAP_DREPLAY(p);

#ifdef DEBUG
fprintf(stderr, "sending Discovery Replay from %d at %.8f\n",
node_adr, Scheduler::instance().clock());
#endif // DEBUG

 sdp->drep_type = TCSAPTYPE_DREP;
 sdp->drep_length = 0x00AA;
 sdp->drep_src_add = node_adr;
 for (int j=0;j<MAX_NEIGHB;j++) sdp->drep_neighbors[j]= drepneighbors[j];
 strcpy(sdp->drep_cert,cert_);
 strcpy(sdp->drep_sign,sign_);

 // ch->uid() = 0;
 ch->ptype() = PT_TCSAP;
 ch->size() = IP_HDR_LEN + sdp->size();
 //ch->iface() = -2;
 //ch->error() = 0;
 //ch->addr_type() = NS_AF_NONE;
 //ch->prev_hop_ = node_adr;
 //ch->direction() = hdr_cmn::DOWN;

 ih->saddr() = node_adr;
 ih->daddr() = IP_BROADCAST;
 ih->sport() = 0; //RT_PORT;
 ih->dport() = 0; //RT_PORT; //ih->sport();
 ih->ttl_ = 1;

 Scheduler::instance().schedule(target_, p, ORDI_SIGN_DELAY);
```

```
}

void
TCSAP::sendIStart(nsaddr_t istrfounders[MAX_NEIGHB]) {
Packet *p          = allocpkt();
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih  = HDR_IP(p);
struct hdr_tcsap_istart     *sist = HDR_TCSAP_ISTART(p);

#ifdef DEBUG
fprintf(stderr, "sending InitStart from %d at %.8f\n",
node_adr, Scheduler::instance().clock());
#endif // DEBUG

 sist->istr_type = TCSAPTYPE_ISTR;
 sist->istr_length = 0x00AA;
 sist->istr_src_add = node_adr;
 for (int j=0;j<kthresh;j++) sist->istr_founders[j]= istrfounders[j];
 strcpy(sist->istr_alfabeta,sign_);
 strcpy(sist->istr_sign,sign_);

 // ch->uid() = 0;
 ch->ptype() = PT_TCSAP;
 ch->size() = IP_HDR_LEN + sist->size();
 //ch->iface() = -2;
 //ch->error() = 0;
 //ch->addr_type() = NS_AF_NONE;
 //ch->prev_hop_ = node_adr;
 //ch->direction() = hdr_cmn::DOWN;

 ih->saddr() = node_adr;
 ih->daddr() = IP_BROADCAST;
 ih->sport() = 0; //RT_PORT;
 ih->dport() = 0; //RT_PORT; //ih->sport();
 ih->ttl_ = 1;

 Scheduler::instance().schedule(target_, p, ORDI_SIGN_DELAY);

}

void
TCSAP::sendIAck(nsaddr_t iackfounders[MAX_NEIGHB]) {
Packet *p          = allocpkt();
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih  = HDR_IP(p);
struct hdr_tcsap_iack     *siak = HDR_TCSAP_IACK(p);


#ifdef DEBUG
fprintf(stderr, "sending InitAck from %d at %.8f\n",node_adr,
Scheduler::instance().clock());
#endif // DEBUG

 siak->iack_type = TCSAPTYPE_IACK;
 siak->iack_length = 0x00AA;
 siak->iack_src_add = node_adr;
 for (int j=0;j<kthresh;j++) siak->iack_founders[j]= iackfounders[j];
 strcpy(siak->iack_cert,cert_);
```

```
 strcpy(siak->iack_alfabeta,sign_);
 strcpy(siak->iack_sign,sign_);

 // ch->uid() = 0;
 ch->ptype() = PT_TCSAP;
 ch->size() = IP_HDR_LEN + siak->size();
 //ch->iface() = -2;
 //ch->error() = 0;
 //ch->addr_type() = NS_AF_NONE;
 //ch->prev_hop_ = node_adr;
 //ch->direction() = hdr_cmn::DOWN;

 ih->saddr() = node_adr;
 ih->daddr() = IP_BROADCAST;
 ih->sport() = 0; //RT_PORT;
 ih->dport() = 0; //RT_PORT; //ih->sport();
 ih->ttl_ = 1;

 Scheduler::instance().schedule(target_, p,
 ORDI_SIGN_DELAY + ORDI_VERIF_DELAY + 2*((kthresh -1)*0.02+ORDI_VERIF_DELAY)
 +2*((kthresh -1)*0.02+THRES_VERIF_DELAY));
}

void
TCSAP::sendDWelcome(nsaddr_t dweldest) {
Packet *p          = allocpkt();
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih  = HDR_IP(p);
struct hdr_tcsap_dwelcome     *sdw = HDR_TCSAP_DWELCOME(p);



 sdw->dwel_type = TCSAPTYPE_DWEL;
 sdw->dwel_length = 0x00AA;
 sdw->dwel_src_add = node_adr;
 strcpy(sdw->dwel_cert,cert_);
 strcpy(sdw->dwel_sign,sign_);

 // ch->uid() = 0;
 ch->ptype() = PT_TCSAP;
 ch->size() = IP_HDR_LEN + sdw->size();
 ch->iface() = -2;
 ch->error() = 0;
 ch->addr_type() = NS_AF_NONE;
 ch->prev_hop_ = node_adr;
 ch->direction() = hdr_cmn::DOWN;

 ih->saddr() = node_adr;
 ih->daddr() = dweldest;
 ih->sport() = 0; //RT_PORT;
 ih->dport() = 0; //RT_PORT; //ih->sport();
 ih->ttl_ = 1;

 Scheduler::instance().schedule(target_, p, ORDI_SIGN_DELAY + ORDI_VERIF_DELAY);

#ifdef DEBUG
fprintf(stderr, "sending Discovery Welcome from %d to %d at %.8f\n",node_adr,
dweldest, Scheduler::instance().clock());
```

```
#endif // DEBUG

}

void
TCSAP::sendCRequest() {
Packet *p          = allocpkt();
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih  = HDR_IP(p);
struct hdr_tcsap_crequest     *scq = HDR_TCSAP_CREQUEST(p);

#ifdef DEBUG
fprintf(stderr, "sending Config Request from %d at %.8f\n", node_adr,
 Scheduler::instance().clock());
fprintf(stderr, "Current Ring : %d\n",curent_ring);
#endif // DEBUG

 scq->creq_type = TCSAPTYPE_CREQ;
 scq->creq_length = 0x00AA;
 scq->creq_src_add = node_adr;
 strcpy(scq->creq_cert,cert_);
 strcpy(scq->creq_sign,sign_);

 // ch->uid() = 0;
 ch->ptype() = PT_TCSAP;
 ch->size() = IP_HDR_LEN + scq->size();
 ch->iface() = -2;
 ch->error() = 0;
 ch->addr_type() = NS_AF_NONE;
 ch->prev_hop_ = node_adr;
 ch->next_hop_ = 0;
 ch->direction() = hdr_cmn::DOWN;

 ih->saddr() = node_adr;
 ih->daddr() = IP_BROADCAST;
 ih->sport() = 0; //RT_PORT;
 ih->dport() = 0; //RT_PORT; //ih->sport();
 ih->ttl_ = curent_ring;

 Scheduler::instance().schedule(target_, p, ORDI_SIGN_DELAY + ORDI_VERIF_DELAY);
}

void
TCSAP::sendCReplay(nsaddr_t crepdest) {
Packet *p          = allocpkt();
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih  = HDR_IP(p);
struct hdr_tcsap_creplay     *scp = HDR_TCSAP_CREPLAY(p);



 scp->crep_type = TCSAPTYPE_CREP;
 scp->crep_length = 0x00AA;
 scp->crep_src_add = node_adr;
 strcpy(scp->crep_cert,cert_);
 strcpy(scp->crep_sign,sign_);

 // ch->uid() = 0;
```

```
 ch->ptype() = PT_TCSAP;
 ch->size() = IP_HDR_LEN + scp->size();
 ch->iface() = -2;
 ch->error() = 0;
 ch->addr_type() = NS_AF_INET;
 ch->prev_hop_ = node_adr;
 ch->next_hop_ = crepdest;
 ch->direction() = hdr_cmn::DOWN;

 ih->saddr() = node_adr;
 ih->daddr() = crepdest;
 ih->sport() = 0;//ipret->dport() = ipret->sport();
 ih->dport() = 0; //ih->sport();
 ih->ttl_ = IP_DEF_TTL;

 Scheduler::instance().schedule(target_, p, ORDI_SIGN_DELAY + ORDI_VERIF_DELAY);

#ifdef DEBUG
fprintf(stderr, "sending Config Replay from %d at %.8f\n", node_adr,
Scheduler::instance().clock());
#endif // DEBUG
}


void
TCSAP::sendCCRequest(nsaddr_t ccreqdest, nsaddr_t ccreqcoallition[MAX_THRESH]) {
Packet *p          = allocpkt();
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih  = HDR_IP(p);
struct hdr_tcsap_ccrequest     *sccq = HDR_TCSAP_CCREQUEST(p);

#ifdef DEBUG
fprintf(stderr, "sending Config Certificate Request from %d at %.8f\n", node_adr,
Scheduler::instance().clock());
#endif // DEBUG

 sccq->ccreq_type = TCSAPTYPE_CCREQ;
 sccq->ccreq_length = 0x00AA;
 sccq->ccreq_src_add = node_adr;
 for (int j=0;j<kthresh;j++) sccq->ccreq_coallition[j]= ccreqcoallition[j];
 for (int j=0;j<2;j++) sccq->ccreq_lowest_hid[j]= 0xAAAAAAAA ;
 strcpy(sccq->ccreq_sign,sign_);

 // ch->uid() = 0;
 ch->ptype() = PT_TCSAP;
 ch->size() = IP_HDR_LEN + sccq->size();
 ch->iface() = -2;
 ch->error() = 0;
 ch->addr_type() = NS_AF_NONE;
 ch->prev_hop_ = node_adr;
 ch->direction() = hdr_cmn::DOWN;

 ih->saddr() = node_adr;
 ih->daddr() = ccreqdest;
 ih->sport() = 0; //RT_PORT;
 ih->dport() = 0; //RT_PORT; //ih->sport();
 ih->ttl_ = IP_DEF_TTL;
```

```
 Scheduler::instance().schedule(target_, p, ORDI_SIGN_DELAY
 + kthresh*ORDI_VERIF_DELAY);
}

void
TCSAP::sendCCReplay(nsaddr_t ccrepdest) {
Packet *p          = allocpkt();
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih  = HDR_IP(p);
struct hdr_tcsap_ccreplay     *sccp = HDR_TCSAP_CCREPLAY(p);



 sccp->ccrep_type = TCSAPTYPE_CCREP;
 sccp->ccrep_length = 0x00AA;
 sccp->ccrep_src_add = node_adr;
 strcpy(sccp->ccrep_thresh_cert,cert_);
 strcpy(sccp->ccrep_sign,sign_);

 // ch->uid() = 0;
 ch->ptype() = PT_TCSAP;
 ch->size() = IP_HDR_LEN + sccp->size();
 ch->iface() = -2;
 ch->error() = 0;
 ch->addr_type() = NS_AF_NONE;
 ch->prev_hop_ = node_adr;
 ch->direction() = hdr_cmn::DOWN;

 ih->saddr() = node_adr;
 ih->daddr() = ccrepdest;
 ih->sport() = 0; //RT_PORT; //ipret->dport() = ipret->sport();
 ih->dport() = 0; //RT_PORT; //ih->sport();
 ih->ttl_ = IP_DEF_TTL;

 Scheduler::instance().schedule(target_, p, ORDI_SIGN_DELAY + ORDI_VERIF_DELAY
 + THRES_SIGN_DELAY);
#ifdef DEBUG
fprintf(stderr, "sending Config Certificate Replay from %d at %.8f\n", node_adr,
 Scheduler::instance().clock());
#endif // DEBUG
}

void
TCSAP::sendCRegister() {
Packet *p          = allocpkt();
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih  = HDR_IP(p);
struct hdr_tcsap_cregister     *scg = HDR_TCSAP_CREGISTER(p);

#ifdef DEBUG
fprintf(stderr, "sending Config Register from %d at %.8f\n", node_adr,
Scheduler::instance().clock());
#endif // DEBUG

 scg->creg_type = TCSAPTYPE_CREG;
 scg->creg_length = 0x00AA;
 scg->creg_src_add = node_adr;
 strcpy(scg->creg_thresh_cert,cert_);
```

```
 strcpy(scg->creg_sign,sign_);

 // ch->uid() = 0;
 ch->ptype() = PT_TCSAP;
 ch->size() = IP_HDR_LEN + scg->size();
 ch->iface() = -2;
 ch->error() = 0;
 ch->addr_type() = NS_AF_NONE;
 ch->prev_hop_ = node_adr;
 ch->direction() = hdr_cmn::DOWN;

 ih->saddr() = node_adr;
 ih->daddr() = IP_BROADCAST;
 ih->sport() = 0; //RT_PORT;
 ih->dport() = 0; //RT_PORT; //ih->sport();
 ih->ttl_ = IP_DEF_TTL;

 Scheduler::instance().schedule(target_, p, ORDI_SIGN_DELAY + ORDI_VERIF_DELAY);

}
```

Annexe B

SCRIPTS DES SIMULATIONS SOUS NS2 (CODE TCL)

```
# ADHOC NETWORK AUTO-INITIALIZATION : SIMULATION WITH TCSAP AGENT

# ======================================================================
# Define options
# ======================================================================
set val(chan)           Channel/WirelessChannel    ;# channel type
set val(prop)           Propagation/TwoRayGround   ;# radio-propagation model
set val(netif)          Phy/WirelessPhy            ;# network interface type
set val(mac)            Mac/802_11                 ;# MAC type
set val(ifq)            Queue/DropTail/PriQueue    ;# interface queue type
set val(ll)             LL                         ;# link layer type
set val(ant)            Antenna/OmniAntenna        ;# antenna model
set val(ifqlen)         50                         ;# max packet in ifq
set val(nn)             5                          ;# number of mobile nodes
set val(rp)             AODV                       ;# routing protocol
#set val(sc)            "mobility_file"
#set val(st)            "trafic_file"
set val(x)              200   ;# X dimension of the topography
set val(y)              200
set val(kthresh)        5
# ======================================================================
# Main Program
# ======================================================================

#
# Initialize Global Variables
#

# create simulator instance

set ns_     [new Simulator]

# setup topography object

set topo    [new Topography]

# create trace object for ns and nam

set tracefd [open sboot1_$val(kthresh)_$val(nn).tr w]
set namtrace    [open sboot1_$val(kthresh)_$val(nn).nam w]
#$ns_ use-newtrace
$ns_ trace-all $tracefd
$ns_ namtrace-all-wireless $namtrace $val(x) $val(y)

# define topology
$topo load_flatgrid $val(x) $val(y)

#
# Create God
#
set god_ [create-god $val(nn)]

#
# define how node should be created
#

#global node setting
```

```
$ns_ node-config -adhocRouting $val(rp) \
                 -llType $val(ll) \
                 -macType $val(mac) \
                 -ifqType $val(ifq) \
                 -ifqLen $val(ifqlen) \
                 -antType $val(ant) \
                 -propType $val(prop) \
                 -phyType $val(netif) \
                 -channelType $val(chan) \
                 -topoInstance $topo \
                 -agentTrace ON \
                 -routerTrace OFF \
                 -macTrace ON

#
#  Create the specified number of nodes [$val(nn)] and "attach" them
#  to the channel.

for {set i 0} {$i < $val(nn) } {incr i} {
    set node_($i) [$ns_ node]
    $node_($i) random-motion 0      ;# disable random motion

}


#puts "Loading scenario file..."
#source $val(st)

#puts "Loading connection pattern..."
#source $val(sc)

$node_(0) set X_ 116.513313407723
$node_(0) set Y_ 170.127409041510
$node_(0) set Z_ 0.000000000000
$node_(1) set X_ 173.373211562081
$node_(1) set Y_ 103.610485035689
$node_(1) set Z_ 0.000000000000
$node_(2) set X_ 156.095657140924
$node_(2) set Y_ 31.178569487562
$node_(2) set Z_ 0.000000000000
$node_(3) set X_ 110.794775861621
$node_(3) set Y_ 10.313164318245
$node_(3) set Z_ 0.000000000000
$node_(4) set X_ 40.232455398969
$node_(4) set Y_ 70.297107114875
$node_(4) set Z_ 0.000000000000



for {set i 0} {$i < $val(nn)} {incr i} {


    $ns_ initial_node_pos $node_($i) 15
    $node_($i) color red
}
```

```
for {set i 1} {$i < $val(nn)} {incr i} {


    $ns_ at 0.0 "$node_($i) color blue";

}

for {set i 0} {$i < $val(nn)} {incr i} {
set tcsap_($i) [new Agent/TCSAP [$node_($i) node-addr]]
$ns_ attach-agent $node_($i) $tcsap_($i)
}

for {set i 0} {$i < $val(nn)} {incr i} {
$tcsap_($i) set state_ 0
$tcsap_($i) set kthresh $val(kthresh)

}



for {set i 0} {$i < $val(nn)} {incr i} {
    $ns_ at 0.0 "$node_($i) color blue";

}



for {set i 0} {$i < $val(nn)} {incr i} {

$ns_ at 0.0 "$tcsap_($i) start"
}

$ns_ at 0.0 "$tcsap_(0) start"



#
for {set i 0} {$i < $val(nn) } {incr i} {
    $ns_ at 60.0 "$node_($i) reset";
}
$ns_ at 60.0 "stop"
$ns_ at 60.01 "puts \"NS EXITING...\" ; $ns_ halt"
proc stop {} {
    global ns_ tracefd
    $ns_ flush-trace
    close $tracefd
}


puts "Starting Simulation..."
$ns_ run
```

```
# ADHOC NETWORK  NODE AUTO-CONFIGURATION : SIMULATION WITH TCSAP AGENT

# ======================================================================
# Define options
# ======================================================================
set val(chan)           Channel/WirelessChannel    ;# channel type
set val(prop)           Propagation/TwoRayGround   ;# radio-propagation model
set val(netif)          Phy/WirelessPhy            ;# network interface type
set val(mac)            Mac/802_11                 ;# MAC type
set val(ifq)            Queue/DropTail/PriQueue    ;# interface queue type
set val(ll)             LL                         ;# link layer type
set val(ant)            Antenna/OmniAntenna        ;# antenna model
set val(ifqlen)         50                         ;# max packet in ifq
set val(nn)             100                        ;# number of mobile nodes
set val(rp)             AODV                       ;# routing protocol
set val(sc)             "mobility_file"
#set val(st)            "trafic_file"
set val(x)              1000   ;# X dimension of the topography
set val(y)              1000
# ======================================================================
# Main Program
# ======================================================================

#
# Initialize Global Variables
#

# create simulator instance

set ns_     [new Simulator]

# setup topography object

set topo    [new Topography]

# create trace object for ns and nam

set tracefd [open s100_1000.tr w]
set namtrace    [open s100_1000.nam w]
$ns_ use-newtrace
$ns_ trace-all $tracefd
$ns_ namtrace-all-wireless $namtrace $val(x) $val(y)

# define topology
$topo load_flatgrid $val(x) $val(y)

#
# Create God
#
set god_ [create-god $val(nn)]

#
# define how node should be created
#

#global node setting
```

```
$ns_ node-config -adhocRouting $val(rp) \
                 -llType $val(ll) \
                 -macType $val(mac) \
                 -ifqType $val(ifq) \
                 -ifqLen $val(ifqlen) \
                 -antType $val(ant) \
                 -propType $val(prop) \
                 -phyType $val(netif) \
                 -channelType $val(chan) \
                 -topoInstance $topo \
                 -agentTrace OFF \
                 -routerTrace OFF \
                 -macTrace ON

#
#  Create the specified number of nodes [$val(nn)] and "attach" them
#  to the channel.

for {set i 0} {$i < $val(nn) } {incr i} {
    set node_($i) [$ns_ node]
    $node_($i) random-motion 0      ;# disable random motion

}


#puts "Loading scenario file..."
#source $val(st)

puts "Loading connection pattern..."
source $val(sc)


for {set i 0} {$i < $val(nn)} {incr i} {

    # 15 defines the node size in nam, must adjust it according to your scenario
    # The function must be called after mobility model is defined

    $ns_ initial_node_pos $node_($i) 15
    $node_($i) color red
}

for {set i 1} {$i < $val(nn)} {incr i} {


    $ns_ at 0.0 "$node_($i) color blue";

}

for {set i 0} {$i < $val(nn)} {incr i} {
set tcsap_($i) [new Agent/TCSAP [$node_($i) node-addr]]
$ns_ attach-agent $node_($i) $tcsap_($i)
}

for {set i 1} {$i < $val(nn)} {incr i} {
$tcsap_($i) set state_ 1
}
```

```
$tcsap_(0) set initial_ring 1
$tcsap_(0) set kthresh 10

for {set i 1} {$i < $val(nn)} {incr i} {
    $ns_ at 0.0 "$node_($i) color blue";

}

$ns_ at 0.0 "$node_(0) color red"

for {set i 1} {$i < $val(nn)} {incr i} {

$ns_ at 0.0 "$tcsap_($i) start"
}

$ns_ at 0.0 "$tcsap_(0) start"

$ns_ at 0.0 "$ns_ set-animation-rate 500.0ms"

#
for {set i 0} {$i < $val(nn) } {incr i} {
    $ns_ at 60.0 "$node_($i) reset";
}
$ns_ at 60.0 "stop"
$ns_ at 60.01 "puts \"NS EXITING...\" ; $ns_ halt"
proc stop {} {
    global ns_ tracefd
    $ns_ flush-trace
    close $tracefd
}


puts "Starting Simulation..."
$ns_ run
```

Résumé

Dans le cadre de la sécurité des réseaux mobiles ad hoc (MANETs), la plupart des recherches jusqu'à présent ont porté principalement sur les modèles de confiance et les problèmes de sécurité de routage, et beaucoup moins d'attention a été accordée à la question de la sécurité d'auto-configuration. Cependant, le manque de sécurité dans la conception de tous les protocoles d'auto-configuration précédemment proposés ouvre la possibilité de nombreuses menaces réelles, en raison des vulnérabilités des réseaux MANETs bien connues et qui sont spécifiques au paradigme ad hoc. En conséquence, cela peut conduire dans des environnements potentiellement hostiles à des attaques sérieuses. Quelques mécanismes ont été proposés pour résoudre ce problème, mais aucun d'eux n'a apporté de solutions satisfaisantes. La sécurité de l'auto-configuration est toujours un problème ouvert. Dans cette thèse, un protocole d'auto-configuration d'adresses IP sans état, robuste et sécurisé pour les réseaux MANETs autonomes est spécifié et évalué sous NS2. Notre solution est basée sur une authentification mutuelle, et un modèle de confiance d'Autorité de Certification et d'Auto-configuration complètement distribué, utilisant le support de la cryptographie à seuil basée sur le problème du logarithme discret (DLP). En déployant un nouveau concept de certificat conjointe liant l'adresse IP à la clé publique, nous résolvons irrévocablement le problème de quelques attaques comme IP spoofing et l'attaque Sybile, sans solution à ce jour par les mécanismes conventionnels. Notre protocole nommé TCSAP (Threshold Cryptography based Secure Auto-configuration Protocol) fournit à la fois la sécurité et la robustesse et surmonte toutes les limites des approches proposées précédemment, tout en assurant l'allocation dynamique des adresses IP en temps opportun.

Mots clés : *TCSAP, Cryptographie à seuil, Logarithme discret , partage de secret, réseaux mobiles adhoc, Auto-configuration, Joint IP address and Public key Certificate, signature partielle ElGamal, NS2.*

Abstract

In the context of Mobile Ad hoc Networks (MANETs) security, most of research so far has been focused primarily on trust models and routing security problems, and much less attention has been given to the auto-configuration security issue. However, the lack of security in the design of all previously proposed auto-configuration schemes opens the possibility of many real threats, due to the well-known MANETs' vulnerabilities which are specific to ad hoc paradigm. This can lead hence to serious attacks in potentially hostile environments. Few schemes have been proposed to solve this issue, but none of them has brought satisfactory solutions. Auto-configuration security is still an open problem. In this thesis, a robust and secure stateful IP address auto-configuration protocol for standalone MANETs is specified and evaluated within NS2. Our solution is based on mutual authentication, and a fully distributed Auto-configuration and Certification Authority trust model, in conjunction with discrete logarithm based threshold cryptography. By deploying a new concept of joint IP address and public key certificate, we solve definitively the problem of some attacks such as IP spoofing and Sybil attacks, unsolved up to now by conventional mechanisms. Our protocol named TCSAP (Threshold Cryptography based Secure Auto-configuration Protocol) provides both security and robustness and overcomes all the limitations of the previously proposed approaches while still ensuring the timely IP address allocation.

Keywords : *TCSAP, Threshold Cryptography, Discrete Logarithm , Secret Sharing, Mobile Ad hoc Networks, Auto-configuration, Joint IP address and Public key Certificate, ElGamal-type threshold signature, NS2.*

ملخص

في سياق أمن شبكات MANETs فإن معظم البحوث ركزت حتى الآن على نماذج الثقة وقضايا أمن التوجيه، وقد تم إعطاء اهتمام أقل بكثير لمسألة أمن التكوين التلقائي. نتيجة لذلك، فإن انعدام الأمن في تصميم جميع مخططات التكوين التلقائي المقترحة سابقا يفتح إمكانية تهديدات حقيقية كثيرة بسبب نقاط الضعف المعروفة لشبكات MANETs . هذا يمكن أن يؤدي بالتالي إلى هجمات خطيرة في بيئات معادية محتملة. وقد اقترحت بعض الآليات لحل هذه المشكلة، ولكن أيا منها لم يعطي حلولا مرضية. أمن التكوين التلقائي يبقى إذا مشكلة مفتوحة. في هذه الأطروحة، نقوم باقتراح بروتوكول جديد مؤمن و قوي للتكوين التلقائي لعناوين IP في شبكات MANETs المستقلة و تقييمه من خلال تجارب تحت NS2. الحل المقترح يعتمد على المصادقة المتبادلة ، وتوزيع بشكل كامل لنموذج الثقة و التكوين التلقائي، وذلك باستخدام ترميز العتبة (cryptographie à seuil) على أساس مشكلة اللوغاريتم الطبيعي (problème du logarithme discret DLP) .عن طريق نشر مفهوم جديد للشهادة المشتركة التي تربط بين العنوان IP و المفتاح العمومي، تمكنا من حل بشكل نهائي بعض الهجمات المستعصية التي لم يتم حلها حتى الآن بالآليات التقليدية. سمينا البروتوكول المقترح TCSAP (Threshold Cryptography based Secure Autoconfiguration Protocol). TCSAP يوفر كلا من الأمن والمتانة ويتغلب على كل القيود المفروضة على المناهج المقترحة سابقا، في الوقت الذي يوفرالتكوين التلقائي لعناوين IP في الوقت المناسب.

الكلمات المفاتيح : *TCSAP*، التكوين التلقائي، MANETs ، ترميز العتبة، اللوغاريتم الطبيعي، الشهادة المشتركة للعنوان IP و المفتاح العمومي، *ElGamal-type threshold signature*، *Threshold Cryptography* ، *NS2*