Numéro d'ordre : 2364
E D S P I C : 615

UNIVERSITÉ BLAISE PASCAL – CLERMONT II

ÉCOLE DOCTORALE SCIENCES
POUR L'INGÉNIEUR DE CLERMONT-FERRAND

Laboratoire d'Informatique, de Modélisation et d'Optimisation des Systèmes

THÈSE

Présentée par

Ismail MANSOUR

pour obtenir le grade de

DOCTEUR D'UNIVERSITÉ

Discipline : Informatique

Contribution à la sécurité des communications des réseaux de capteurs sans fil

Soutenue publiquement le 5 juillet 2013 devant le jury :

Rapporteurs :

M. Thierry VAL, *Professeur à l'université de Toulouse II, Toulouse (Président)*

M. Pascal LAFOURCADE, *MCF à l'université Joseph Fourier, Grenoble*

Examinateurs :

M. Alain QUILLIOT, *Professeur à l'université Blaise Pascal, Clermont-FD (Directeur de thèse)*

M. Michel MISSON, *Professeur à l'université d'Auvergne, Clermont-FD (Directeur de thèse)*

M. Gérard CHALHOUB, *MCF à l'université d'Auvergne, Clermont-FD (Co-encadrant)*

M. Mohamad BADRA, *Chercheur CNRS au laboratoire LIMOS, Clermont-FD (Co-encadrant)*

M. Adrien VAN DEN BOSSCHE, *MCF à l'université de Toulouse II, Toulouse*

Invité :

M. Badr RMILI, *Ingénieur CNES, Paris*

À la mémoire de mon père…

Remerciements

J'exprime mes sincères remerciements et toute ma gratitude, en premier lieu, à M. Michel MISSON et M. Gérard CHALHOUB qui ont dirigé et encadré cette thèse. C'est grâce à leurs expertises, leurs conseils et leur écoute que ce travail a pu aboutir. Je les remercie aussi pour le temps considérable qu'ils ont passé à relire et corriger le manuscrit de thèse, les articles et les présentations. J'aimerais aussi souligner que leur travail ne se limite pas à l'aspect scientifique mais aussi humain et relationnel. Je pense à la gentillesse touchante de M. MISSON et à la l'enthousiasme communicatif et l'énergie débordante de Gérard qui m'a encadré de près. Il a su guider mon travail avec intelligence et a fait preuve de disponibilité en toute circonstance.

Ma gratitude va également vers M. Alain QUILLIIOT et M. Mohamad BADRA qui m'ont accueilli et dirigé pendant les premières étapes de la thèse. Je les remercie pour leur accompagnement et leur écoute au cours de ces années.

Je tiens à remercier très chaleureusement tous les membres du Jury. Je pense particulièrement aux rapporteurs M. Thierry VAL et M. Pascal LAFOURCADE qui m'ont fait l'honneur d'accepter d'évaluer ce travail. Les échanges que j'ai eus avec eux m'ont beaucoup enrichi. Je n'oublie pas de remercier M. Adrien VAN DEN BOSSCHE d'avoir examiné mon manuscrit et M. Badr RMILI pour l'intérêt qu'il a manifesté pour ce travail.

Merci également à tous les membres de l'équipe réseaux et protocoles du laboratoire LIMOS, pour leurs conseils, encouragements, et pour la sympathie qu'ils m'ont témoignée. Je remercie en particulier M. François DELOBEL pour sa collaboration dans la réalisation du maquettage de notre contribution.

Je tiens à remercier également M. Joël TOUSSAINT et Mme Anne-Sophie CAPELLE-LAIZÉ qui m'ont accompagné durant mon service d'enseignement à l'IUT de Clermont-FD et à l'IUT de Poitiers.

Mes derniers remerciements vont à ma famille au Liban et à tous mes amis libanais, un peu partout en France et particulièrement à Clermont-FD, avec qui j'ai partagé des moments inoubliables. J'avais de la chance de les avoir à mes côtés pendant les moments difficiles.

Table des matières

Table des figures

Glossaire

Introduction

Ce travail a été réalisé au sein de l'équipe Réseaux et Protocoles [1] du laboratoire LIMOS [2] de l'Université Blaise Pascal de Clermont-Ferrand [3]. Il a été financé par le FEDER (Fonds européen de développement régional) [4], la région d'Auvergne, l'Université Blaise Pascal et le laboratoire LIMOS.

Notre contribution a été élaborée en trois étapes. La première étape, réalisée sous la direction de M. Alain QUILLIOT et l'encadrement de M. Mohamad BADRA, consistait à étudier les protocoles et les systèmes cryptographiques des réseaux filaires. Cette étude nous a emmenés lors d'une deuxième étape à identifier les algorithmes et les primitives cryptographiques adaptables aux réseaux de capteurs sans fil. La troisième étape de ce travail a été consacrée à proposer, implémenter et évaluer une nouvelle architecture de sécurité pour les réseaux de capteurs sans fil en se basant sur les deux premières étapes. La deuxième et la troisième étape ont été réalisées sous la direction de M. Michel MISSON et l'encadrement de M. Gérard CHALHOUB.

Un réseau de capteurs sans fil (RCSF) est un ensemble de nœuds généralement dédiés à la collecte d'information, capables de communiquer entre eux afin de réaliser des tâches diverses. La facilité de déploiement de ce type de réseau constitue un atout qui les rend facilement intégrables dans une grande variété d'applications comme les applications militaires, environnementales, domotiques, industrielles, etc. Ses nœuds étant d'une petite taille, souvent de l'ordre d'une pièce de monnaie, cela permet leur déploiement dans des endroits à accès difficile ou dangereux pour l'être humain. Les RCSF représentent une nouvelle perspective pour un grand nombre d'applications qui concernent l'environnement. Ils sont essentiellement utilisés pour la récolte des données comme par exemple celles de phénomènes physiques, de la surveillance des évolutions de terrains, etc.

Du fait de la miniaturisation et de la production à grande échelle et à faible coût, les nœuds d'un RCSF sont limités en capacité de calcul et de mémoire ainsi qu'en ressources énergétiques. Ces contraintes représentent un véritable challenge pour les chercheurs quand il s'agit de concevoir des solutions répondant aux besoins que nous venons d'évoquer. Un des objectifs les plus essentiels est l'économie de l'énergie des nœuds qui a pour but de prolonger la durée de vie d'un tel réseau. Les protocoles nécessaires au fonctionnement des RCSF

comme les protocoles d'accès au médium, de routage et de sécurité doivent prendre en compte les contraintes des nœuds du réseau tout en économisant le plus possible leur consommation d'énergie. Le but des nouvelles propositions et des nouveaux protocoles dédiés aux RCSF est de faire un compromis entre la qualité du service rendu par ces solutions et le respect des contraintes imposées par les nœuds.

Il est connu que les RCSF sont faciles à attaquer en raison de la nature du médium qui permet relativement facilement aux adversaires d'écouter, de falsifier ou d'injecter des données dans le réseau. Si les nœuds capteurs sont contraints par des capacités limitées, il n'en est pas de même pour les adversaires qui n'utilisent pas nécessairement la même technologie pour lancer leurs attaques. Ainsi, les besoins de sécuriser les communications d'un RCSF sont en croissance ces dernières années et varient d'un domaine applicatif à un autre. Parmi les applications qui demandent un grand niveau d'authentification et de confidentialité de communications, nous retrouvons celles des sites industriels et militaires, tandis que d'autres, comme les applications environnementales, demandent plutôt l'intégrité et la fiabilité des communications.

Des efforts ont été faits ces dernières années afin d'adapter des mécanismes et des algorithmes cryptographiques dans le but de sécuriser les communications des RCSF. Parmi ces efforts, nous retrouvons des standards connus comme ZigBee [5], WirelessHART [6] et ISA100.11a [7]. Le but est de proposer des solutions de sécurité qui permettent d'offrir, d'une part, la confidentialité, l'intégrité et l'authentification des données, et d'autre part, l'authentification des nœuds du réseau. Le niveau de sécurité variant d'une application à une autre selon l'importance des informations échangées, les solutions doivent couvrir la plupart des services avec un coût acceptable en consommation d'énergie, de mémoire et de temps de calcul.

Notre contribution est destinée aux applications utilisant les RCSF ayant de nœuds avec une faible mobilité et qui demandent un niveau de sécurité élevé. Contrairement à la plupart des méthodes de la littérature destinées à des topologies spécifiques, notre proposition de sécurité peut couvrir à la fois les besoins des topologies plates et hiérarchiques. En supposant l'existence d'une station de base dans les deux topologies comme la seule destination de données (le puits), nous avons proposé une méthode d'association dynamique des nœuds qui repose sur l'utilisation de cette station comme la seule entité de confiance du réseau. Notre proposition est dynamique du fait qu'elle permet aux nouveaux nœuds de

rejoindre le réseau à tout moment. Pour ce faire, il suffit seulement de stocker trois clés asymétriques dans un nouveau nœud avant son déploiement, et de le déclarer au niveau de la station de base. Nous avons utilisé des mécanismes cryptographiques asymétriques basés sur les courbes elliptiques permettant le partage secret de clés symétriques entre les nœuds. Ces dernières sont dédiées au chiffrement de données lors de la période du déploiement. Nous proposons la méthode DHWI (*Diffie-Hellman Without Interaction*) qui offre plus de sécurité que la méthode classique d'échange de clés de *Diffie-Hellman*. Elle permet d'économiser de l'énergie en supprimant les échanges préliminaires aboutissant à la création de la clé commune entre un nœud et la station de base. Une fois le réseau déployé et sécurisé, grâce à la méthode d'association au réseau, nous proposons deux méthodes complémentaires afin de maintenir le niveau de sécurité attendu. La première méthode permet à un nœud de découvrir son voisinage afin de créer plusieurs chemins sécurisés vers la station de base. La deuxième méthode permet de révoquer et de renouveler les clés distribuées en passant par la station de base. Nos évaluations et mesures réelles sur les cartes TelosB/TinyOS, donnent une idée précise et détaillée du surcoût de la consommation de ressources nécessaire pour assurer le niveau élevé de sécurité attendu.

Dans le premier chapitre, nous commençons par une introduction des RCSF en citant quelques domaines d'applications et en montrant les besoins de sécurité au niveau de leurs communications. Nous insistons sur les services essentiels qui permettent à une communication d'être sécurisée. Nous présentons enfin les topologies auxquelles nous avons dédié notre proposition de sécurité.

Le deuxième chapitre débute par l'étude des attaques qui concernent les RCSF, en soulignant les remèdes employés pour résister à chacune d'entre-elles. Nous présentons ensuite les services offerts par les systèmes de cryptographie pour protéger les communications des réseaux traditionnels câblés. Nous montrons enfin leurs adaptations dans les RCSF en étudiant les solutions de la littérature en termes d'algorithmes et de gestion de clés.

Dans le troisième chapitre, nous présentons notre proposition d'architecture dynamique de sécurité pour les RCSF. Nous évaluons notre proposition en termes de résistance aux attaques et nous la comparons à d'autres méthodes existantes.

Dans le quatrième chapitre, nous commençons d'abord par présenter l'implémentation de la solution. Ensuite, nous présentons l'évaluation de notre proposition en termes

d’occupation mémoire ainsi que le temps d’exécution et l’énergie consommés par les nœuds. Nous finissons ce chapitre par une comparaison de notre proposition avec les méthodes existantes en termes d’utilisation de ressources.

Ce mémoire de thèse se termine par une conclusion générale et un ensemble de perspectives. La perspective principale consiste à utiliser la technique de saut de fréquences afin de rendre notre proposition plus robuste aux attaques spécifiques, comme le *Jamming*.

Chapitre 1

Les réseaux de capteurs sans fil

Ces dernières années, les réseaux de capteurs sans fil (RCSF) ont attiré l'attention des chercheurs et des services de R&D en raison de leur simplicité de déploiement et de leur potentiel applicatif dans des domaines très divers. Parmi ceux-ci nous citerons les domaines militaires, environnementaux, industriels, etc. Les RCSF sont essentiellement utilisés pour la récolte des données pour les envoyer à des stations de base, des unités de traitement ou des serveurs de stockage.

Un RCSF est un ensemble de nœuds, appelés capteurs, limités en capacité mémoire et de calcul, devant être économes en énergie, ce qui les contraint à exploiter une faible puissance de transmission et des portées et des débits modestes. Ces capteurs servent à récolter des données comme une détection de présence, un relevé de température ou d'humidité, etc. Les données récoltées par les nœuds, alors appelés nœuds sources, sont généralement envoyées vers une ou plusieurs destinations selon la nature de l'application supportée. Dans un grand nombre de cas ces données sont centralisées en un point de collecte unique appelé puits qui peut servir aussi de relai pour atteindre une destination plus lointaine. Dans ce cas le puits est aussi désigné par passerelle ou station de base. La faible portée implique bien souvent d'avoir recours au routage multi-sauts pour faire cheminer l'information. Une information produite par un nœud source est transmise à travers un chemin de nœuds commençant par l'un de ses voisins et terminant par le nœud destination (souvent le puits). Ce dernier est généralement connecté en filaire ou en sans fil à une unité (ou hôte) de contrôle dédiée aux traitements des données récoltées par les capteurs et à la gestion du RCSF. Dans la Figure I.1, nous présentons un exemple d'applications utilisant un RCSF. Les nœuds du RCSF collectent les données et les envoient à la station de base. Cette dernière transmet les données reçues vers un hôte de contrôle via un réseau filaire ou sans fil.

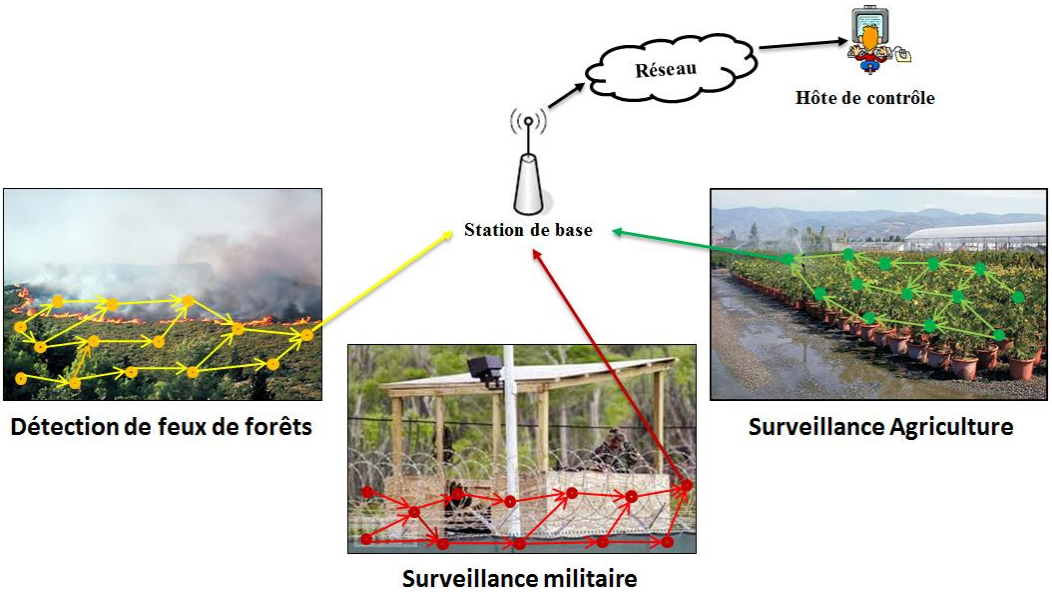


Figure I.1 : Exemple d'applications utilisant un RCSF.

Le fait d'utiliser l'air pour échanger les informations rend la sécurisation des communications d'un RCSF plus difficile que celle connue des réseaux câblés. En effet, un adversaire pourra, par exemple, facilement injecter des données erronées pour perturber le trafic. Ceci pourra causer une saturation au niveau des communications. Ainsi, l'application risque de tomber dans un déni de service et ne pourra plus répondre aux besoins pour lesquels le réseau a été déployé.

Les RCSF partagent avec les MANET *(Mobile Ad hoc NETworks)* plusieurs propriétés. La ressemblance que nous pouvons voir immédiatement est que les deux sont des réseaux sans fil distribués avec une infrastructure pas toujours bien définie. Des chercheurs dans le domaine des MANET se sont consacrés à cette ressemblance, mais aussi à d'autres similarités comme la limitation d'énergie, de mémoire et de capacité de calcul. Ils essaient d'apporter leur savoir-faire et d'adapter aux RCSF les algorithmes, protocoles et techniques dédiés initialement aux MANET. Ces deux classes de réseaux sans fil se différencient généralement par le domaine des applications qui leur est associé. Les MANET sont plutôt proches de l'être humain dans le sens où l'interaction de ce dernier est très souvent directe avec les nœuds du réseau : PC portables, terminaux mobiles, téléphones, etc. Tandis que l'interaction avec un nœud des RCSF est avec l'environnement ; le réseau est essentiellement utilisé pour la récolte des données de phénomènes physiques. Le déploiement d'un RCSF pourra être critique lorsque les zones sont vastes et parfois hostiles. Les nœuds capteurs de ces réseaux deviennent alors fragiles et assez vulnérables à diverses formes de défaillance : capture

physique, épuisement d'énergie, etc., Le niveau de sécurité requis pouvant différer selon l'application, il est donc nécessaire de proposer des solutions adaptées pour chacune d'entre-elles selon le niveau de sécurité attendu.

Nous abordons d'abord dans ce chapitre les domaines d'applications des RCSF. Ensuite, nous présentons les besoins de sécurité des applications qui utilisent les RCSF et nous montrons comment ils doivent être assurés. Et finalement, nous introduisons les deux topologies les plus connues des RCSF et pour lesquelles nous avons proposé notre architecture de sécurité.

I.1 Domaines d'applications et besoins de sécurité

Les RCSF se révèlent très utiles dans de nombreux domaines d'application [8]. Plusieurs parmi eux ont attiré l'attention et ont fait l'objet d'une recherche et d'un développement universitaire ou industriel. Nous pouvons citer parmi eux les applications militaires, environnementales, médicales, industrielles, etc. Nous décrivons brièvement quelques applications pour donner une idée sur le niveau de sécurité attendu par chacune d'elles.

I.1.1 Applications militaires

Comme la plupart des avancées technologiques de l'information, les RCSF peuvent être utiles voire nécessaires pour des applications militaires stratégiques. La DARPA [9] par exemple a cru très tôt à cette nouvelle sorte de réseaux et a soutenu des projets qui ont notamment conduit à l'émergence de plateformes de référence pour la recherche académique : Telos B, TinyOS. Les RCSF peuvent être rapidement déployés et utilisés pour la surveillance des champs de bataille afin de fournir des informations extrêmement importantes comme le nombre, la disposition et les mouvements des soldats adversaires [10]. Ces réseaux peuvent être déployés pour couvrir des terrains difficiles d'accès ou bien pour analyser des champs en termes de présence d'agents chimiques, biologiques ou nucléaires avant d'envoyer les soldats.

Les informations collectées et envoyées vers la station de base ont besoin d'être assurées en termes de confidentialité, intégrité et authentification de leurs sources expéditrices. En effet, si le camp adversaire peut décrypter ou modifier les communications

du réseau déployé, il peut alors rapidement s'en servir à son avantage pour leurrer les troupes ennemies en fabriquant des pièges par exemple.

I.1.2 Applications environnementales

Les RCSF peuvent être déployés dans la nature [11] pour des raisons liées à la : (i) protection de l'environnement, comme la détection de feux de forêt pour signaler les incendies, (ii) progression de certains domaines, comme dans le cas de détection de zones sèches dans les champs agricoles afin de rendre l'irrigation plus efficace et, (iii) surveillance des sites sensibles, comme dans le cas des sites nucléaires ou industriels où nous avons besoin d'une surveillance continue pour détecter des fuites des éléments radioactifs ou de produits toxiques.

Les applications environnementales ont des besoins différents pour sécuriser leurs communications. Par exemple, quand les capteurs déployés dans une forêt envoient la prise régulière de température de certaines régions de la forêt, nous avons besoin de protéger l'intégrité de cette température et authentifier sa source. Un adversaire peut voir cette température mais il ne doit pas être capable de créer de fausses alertes ou masquer des alertes réelles.

I.1.3 Applications industrielles

Les applications industrielles [12] peuvent désigner des activités liées à la production de produits comprenant un suivi et un contrôle. Les RCSF offrent pour ces applications une flexibilité qui permet de franchir les contraintes liées aux câblages. Par exemple, dans une usine de traitement chimique à plusieurs étapes, il peut y avoir des capteurs placés en différents points pour surveiller la concentration chimique, la pression, etc. Les informations de cette surveillance doivent être en temps réel car elles peuvent être utilisées à des fins d'asservissement : pour ajuster les quantités d'ingrédients ou modifier les températures par exemple.

Le besoin de sécurité des communications de ces applications repose sur l'utilisation sûre, confidentielle et fiable de données. Si un concurrent mal intentionné (espionnage industriel) récupère ou modifie les informations liées à une application chimique par exemple, cela peut entraîner de graves conséquences suite à l'ajout d'ingrédients toxiques qui peuvent causer des pollutions ou des explosions par exemple.

I.2 Services pour une communication sécurisée

Comme nous l'avons souligné dans la partie précédente, le niveau de sécurité des communications peut différer d'une application à une autre. Cependant, la sécurité de comunications de n'importe quelle application utilisant le RCSF commence par la protection des liens entre chaque paire de noeuds de ce réseau. Elle pourra être assurée en utilisant la cryptographie des échanges (c'est-à-dire l'usage du chiffrement, déchiffrement, de la signature, etc.) d'une part, et de la gestion des clés de chiffrement d'autre part. La complexité de l'algorithme de chiffrement, la taille des clés et la gestion de ces clés peuvent varier d'une application à une autre selon le niveau de sécurité attendu. Parmi les services qui doivent être assurés pour une communication sécurisée [13], nous citons :

- ***La confidentialité :*** elle consiste à s'assurer que personne ne pourra interférer dans la communication entre deux nœuds afin de récupérer les données lors de leur transfert. Les algorithmes cryptographiques de nos jours peuvent assurer une confidentialité suffisante en utilisant des clés de grande taille. Cependant, ces algorithmes sont lourds en termes de mémoire et de calcul, de plus pour les RCSF il faut considérer le temps et l'énergie qu'ils consomment. Il est nécessaire d'adapter ces algorithmes pour qu'ils puissent être un bénéfice et non pas un handicap pour les RCSF.
- ***L'intégrité des données :*** c'est le fait de s'assurer que les données ne sont pas modifiées lors de leur transfert dans le réseau. Afin de garantir ce service, il faut associer à ces données une redondance codée qui atteste la validité de ce qui est reçu avec une plus grande probabilité que celle d'un CRC (*Cyclic Redundancy Check,* comme le fait CRC pour détecter les erreurs de transmissions). Ceci peut être réalisé en utilisant des fonctions de hachages. Elles permettent de calculer l'empreinte des données. Ensuite, les données sont envoyées avec leur empreinte. Le destinataire pourra vérifier l'intégrité des données en calculant de nouveau l'empreinte des données reçues et en la comparant avec celle qui a été envoyée. Notons que les empreintes sont signées (avec la clé privée de l'expéditeur) avant d'être envoyées ; cela est fait pour garantir l'authentification de la source.
- ***L'authentification d'un nœud :*** c'est le fait de pouvoir vérifier et connaître avec une grande certitude l'identité d'un nœud. Elle pourra être assurée implicitement dans un système à chiffrement symétrique, car dans un tel système, deux entités partagent d'une façon sûre une seule clé secrète. Alors, seule l'entité qui possède la clé secrète

pourra chiffrer/déchiffrer les données. Dans les systèmes asymétriques, l'authentification pourra être assurée lors de l'utilisation d'un certificat d'identification. C'est une relation sûre entre la clé publique et l'identité de l'entité. Le destinataire doit être capable d'identifier l'expéditeur en contactant un tiers de confiance qui a la responsabilité de signer, délivrer et vérifier la validité des certificats.

- ***La non-répudiation :*** c'est le fait qu'aucun nœud ne pourra nier (désavouer) le fait d'avoir échanger des informations. Autrement dit, la non-répudiation de l'origine prouve que les données ont été envoyées, et la non-répudiation de l'arrivée prouve qu'elles ont été reçues. Nous citons quelques techniques utilisées dans les réseaux filaires pour assurer la non-répudiation :
    - preuve temporelle : Quand un serveur reçoit un message envoyé de la part d'un client, il pourra ajouter une date de réception au message avant de calculer son empreinte. Ensuite, il signe l'empreinte avec sa clé privée avant de l'envoyer au client. Ce dernier possède maintenant une preuve de l'heure à laquelle ce message a été reçu.
    - preuve d'historique : Les protocoles standards comme TLS *(Transport Layer Security)* [14] utilisent le chiffrement des discussions antérieures. Durant l'établissement d'un lien entre deux entités et à la fin de chaque nouvelle session sécurisée TLS, les deux entités chiffrent leur historique d'échange avant de l'envoyer à leur partenaire.

Les principaux services cités ci-dessus sont assurés en utilisant les protocoles standards des réseaux traditionnels (comme TLS, IPsec *(Internet Protocol Security)* [15] , etc.). Ces protocoles n'ont pas été conçus pour être supportés par des entités de capacité mémoire et de calcul optimisé, ils ne peuvent pas donc être appliqués directement au RCSF. Cependant, des travaux de recherche ont été publiés récemment afin d'adapter des parties de ces protocoles pour les RCSF [16], [17].

Notre contribution a pour but d'assurer ces services (ci-dessus) en respenctant les limites matérielles et logicielles des nœuds d'un RCSF. Elle concerne les communications qui passent via le RCSF comme le montre la Figure I.2. Les données sont générées par un processus d'applications et passent via le RCSF qui les routent jusqu'à leur arrivée à la station de base. Cette dernière joue le rôle d'une passerelle servant à transférer les informations à travers un réseau filaire ou sans fil traditionnel. Les utilisateurs communiquent avec le

processus en utilisant un intergiciel *(middleware)* et stockent les données dans des serveurs appropriés. Nous utilisons des mécanismes de chiffrement asymétrique existants basés sur les courbes elliptiques afin d'échanger des clés symétriques pour assurer le chiffrement de ces données. Notre proposition de sécurité est destinée à des applications avec une faible mobilité où le niveau de sécurité attendu est élevé.

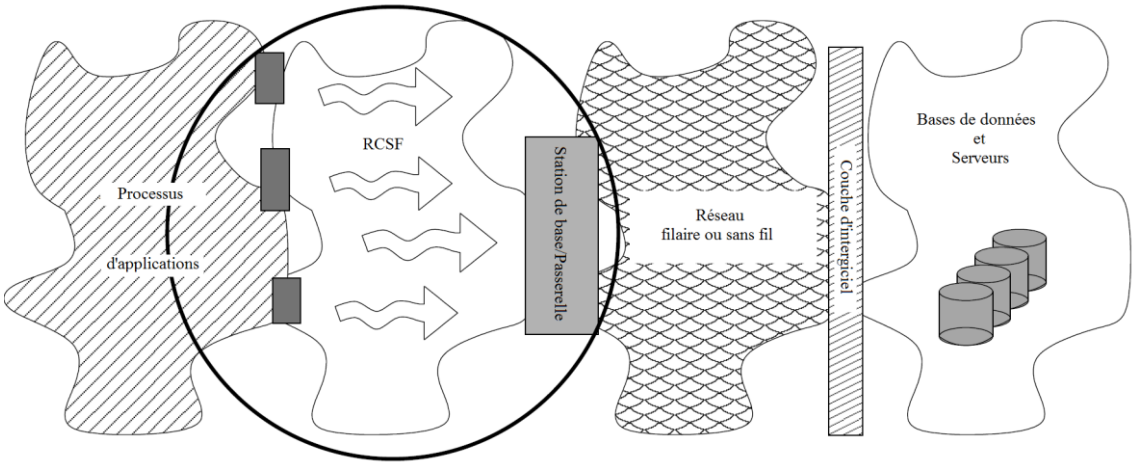


Figure I.2 : Chaîne d'acquisition de données incluant des RCSF.

I.3 Topologies de RCSF

Indépendamment des topologies physiques, nous avons choisi de considérer deux topologies représentatives qui se distinguent par le rôle attribué aux nœuds : plate et hiérarchique. Nous introduisons ici les caractéristiques générales de ces deux topologies alors que nous consacrons la partie III.1.1.1 à l'étude de leurs besoins en termes de liens sécurisés pour protéger leurs communications.

I.3.1 Topologie Plate

Dans la topologie plate d'un RCSF, l'ensemble des nœuds capteurs ont le même rôle et sont reliés entre eux dans le but de créer des chemins qui atteignent la station de base. La Figure I.3 montre un exemple d'une topologie plate d'un RCSF. Le nœud S désigne la station de base et les nœuds de couleur gris foncé désignent des capteurs. Les données sont routées d'un capteur à un autre afin d'arriver à la destination S.

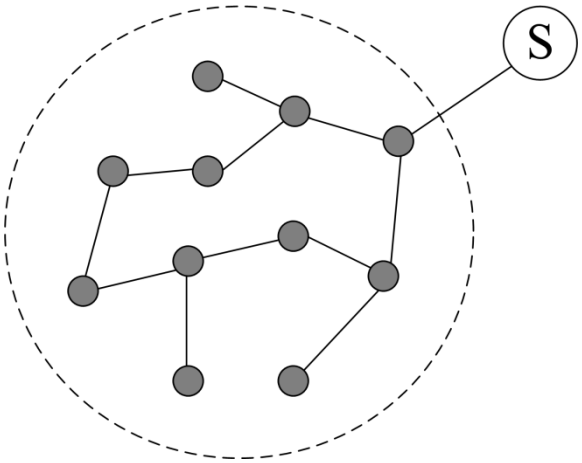


Figure I.3 : Exemple d'une topologie plate d'un RCSF.

I.3.2 Topologie Hiérarchique

Dans la topologie hiérarchique d'un RCSF, tous les nœuds capteurs n'ont pas le même rôle. La topologie est organisée à partir d'un ensemble de *clusters* reliés entre eux grâce à des nœuds ayant des fonctionnalités de routeurs. Ces chefs de *clusters* ou *clusterheads* sont chargés de router les informations collectées par leurs nœuds capteurs vers la station de base. La Figure I.4 montre un exemple d'une topologie hiérarchique d'un RCSF. Le nœud S désigne la station de base, les nœuds R désignent des routeurs et les nœuds de couleur gris foncé désignent des capteurs. Les données récoltées par un nœud sont envoyées à son chef de *cluster* qui, à son tour, les envoient en exploitant d'une façon ascendante la hiérarchie des clusters, pour atteindre la station de base.

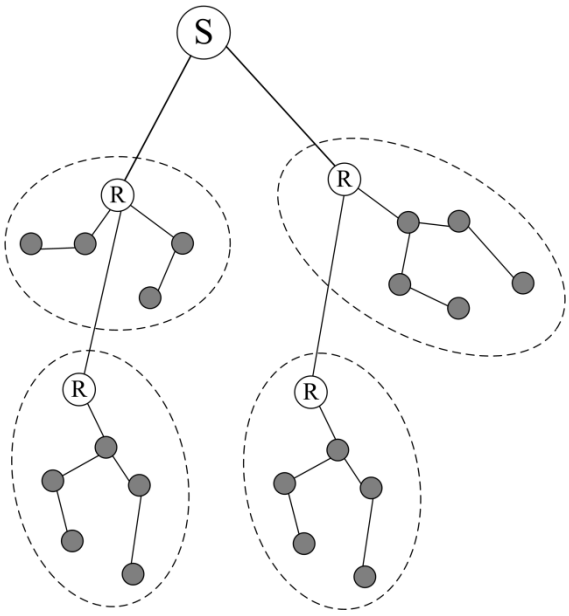


Figure I.4 : Exemple d'une topologie hiérarchique d'un RCSF

I.4 Conclusion

Les RCSF présentent un potentiel applicatif dans des domaines très divers. Nous avons cités dans ce chapitre certains de ces domaines en soulignant leurs différents besoins pour sécuriser leurs communications.

Pour notre proposition, nous avons considéré deux topologies représentatives des RCSF : plate et hiérarchique. Elles se distinguent par le rôle attribué aux nœuds. Dans une topologie plate, l'ensemble des nœuds capteurs ont le même rôle et sont reliés entre eux afin d'atteindre la station de base. Tandis que dans une topologie hiérarchique les nœuds capteurs ont un rôle différent. Ils sont reliés entre eux grâce à des nœuds ayant des fonctionnalités de routeurs. Ces derniers sont chargés de router les informations collectées par leurs nœuds capteurs vers la station de base. Les exemples de figures de ces deux topologies (Figure I.3 et Figure I.4) contribuent à l'élaboration de notre étude de liaisons possibles dans la partie III.1.1.1, et de notre représentation de branches de la Figure IV.9 utilisée dans l'évaluation de notre proposition.

Chapitre 2

Sécurité des communications dans les RCSF

Dans ce chapitre, nous commençons par présenter les attaques et les contremesures dans les RCSF. Ensuite, nous abordons la cryptographie et ses systèmes utilisés dans les réseaux filaires pour protéger les communications. Puis nous montrons leurs principales adaptations nécessaires pour les utiliser dans les RCSF, qui sont décrites dans la littérature, notamment en termes de gestion et de distribution de clés. Nous concluons ce chapitre en résumant ce qui a été proposé pour sécuriser les communications dans les RCSF.

II.1 Attaques et contremesures

La plupart des attaques détaillées dans cette partie concernent les réseaux sans fil comme WLAN, MANET et WSN. La vulnérabilité commune entre ces différents types de réseaux est l'utilisation d'une communication sans fil. Notre travail bibliographique a montré qu'il était d'usage de classifier les attaques en les divisant en catégories selon des critères ou des facteurs bien définis. L'entité susceptible de porter une attaque sera désignée par l'adversaire dans la suite. Nous citons les classifications les plus connues :

- Protocole de communication : les attaques sont distinguées selon la couche protocolaire du modèle OSI [18] et [19].
- Intention de l'adversaire : les attaques sont classées selon les intentions primaires des adversaires comme la collecte des informations, la perturbation des communications ou l'agrégation des données, l'épuisement des ressources et la capture physique de nœuds [20].
- Attaques passives/actives : une attaque passive est une attaque qui vise à écouter et collecter les données qui pourront être utilisées plus tard pour

commencer d'autres types d'attaques. Tandis que l'attaque active cherche à modifier, fabriquer ou perturber les données [21], [22] et [23].

- Attaques internes/externes : quand un nœud appartenant au réseau participe à l'attaque, cette dernière sera considérée comme une attaque interne. Elle pourra être réalisée en utilisant des nœuds du réseau qui ont été capturés physiquement. Tandis que si le nœud participant à l'attaque est extérieur au réseau, l'attaque sera considérée comme une attaque externe [23].
- Lieu de l'attaque : les auteurs dans [24] définissent un nouveau modèle d'attaques approprié aux RCSF, basé sur la cible de l'attaque. Ce modèle se distingue en fonction de « où » et « sur quelle » entité du réseau les attaques sont effectuées (*i.e.*, station de base, voisin, source de l'attaque).

Chacune de ces méthodes de classification aide à voir les menaces selon différentes perspectives. Nous avons illustré dans la Figure II.1 les attaques les plus connues dans les RCSF selon deux classifications parmi celles citées ci-dessus. La partie gauche de la figure présente une classification selon les couches. La partie droite de la figure divise les attaques selon l'intention d'un adversaire. Comme la figure le montre, les attaques peuvent se dérouler à divers endroits du réseau pour des résultats différents et avoir un impact sur plusieurs couches. Nous remarquons que l'attaque Sybil (voir partie II.1.3.1) a pu se placer entre deux couches.

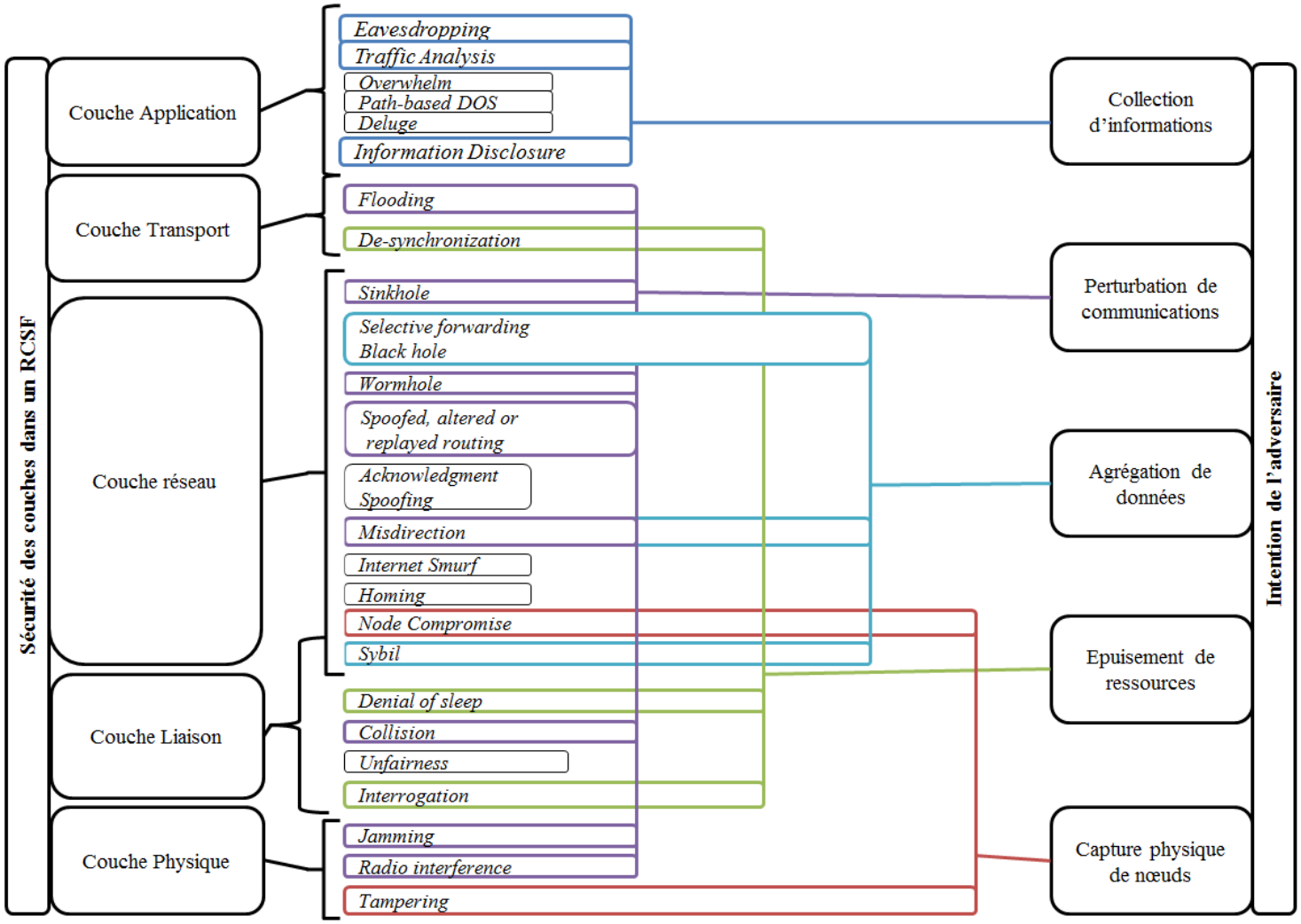


Figure II.1 : Classification des attaques vue selon deux perspectives : couches et intention de l'adversaire.

Nous détaillons dans la suite quelques attaques et les contremesures pour se défendre de leurs effets. Nous avons choisi de répartir les attaques suivant l'intention de l'adversaire.

II.1.1 Collection d'informations

L'adversaire commence à collecter et analyser les données grâce à des attaques du genre « collection d'informations ». Il peut par la suite utiliser ces données pour déclencher d'autres types d'attaques selon les failles trouvées par ses analyses.

II.1.1.1 *Eavesdropping* ou *Passive Monitoring*

L'attaque *Eavesdropping* fait partie des attaques passives pour lesquelles les adversaires cherchent à surveiller ou à collecter les informations circulant dans le réseau. Le but de cette attaque est d'écouter le trafic sur les canaux de communication et d'intercepter les paquets. Tout dépend des mécanismes utilisés pour sécuriser les échanges. Si ces derniers ne sont pas chiffrés, l'adversaire pourra alors récupérer immédiatement leur contenu. Dans le cas contraire, l'adversaire sera obligé d'utiliser d'autres types d'attaques comme *Information Disclosure* afin de récupérer les contenus des paquets.

II.1.1.2 *Traffic Analysis*

Cette attaque est utilisée pour connaître un maximum d'informations sur le RCSF comme la topologie, le protocole MAC, le protocole de routage et les mécanismes de sécurité comme l'authentification et les algorithmes de chiffrement, etc. L'analyse des paquets reçus ou envoyés d'un nœud pourra donner des précisions sur son rôle. Par exemple, les nœuds qui reçoivent/envoient beaucoup de paquets sont considérés comme des nœuds actifs et peuvent être des routeurs, des agrégateurs de données ou des distributeurs de clés de chiffrement.

II.1.1.3 *Information Disclosure*

Cette attaque consiste à révéler les informations cachées dans les paquets. Si les données ne sont pas bien protégées par un bon moyen de chiffrement, elles peuvent être récupérées facilement par l'adversaire à l'aide d'une simple analyse. Il s'agit de l'attaque par force brute : une méthode qui teste, une par une, toutes les combinaisons possibles d'une clé. Pour contrer cette attaque, il faut utiliser des clés suffisamment grandes ou renouvelées périodiquement.

II.1.2 Perturbation des communications

Nous détaillons dans la suite plusieurs méthodes pour perturber les communications dans les RCSF. Les attaques de cette catégorie sont considérées comme des attaques actives qui visent les couches : physique, liaison de données, réseau et transport de la pile protocolaire.

II.1.2.1 *Jamming*

Un adversaire identifie les fréquences radio utilisées par le RCSF visé et essaie de perturber ou de bloquer ses communications. Il s'agit de mettre une antenne pour émettre des signaux sur les mêmes fréquences afin d'empêcher les nœuds de communiquer sur ces fréquences. Cette attaque s'appelle le *Jamming*, elle fait partie des attaques qui causent un déni-de-service. La Figure II.2 illustre ce type d'attaque. L'adversaire place son antenne proche des nœuds ciblés par l'attaque. La plupart des adversaires préfèrent attaquer une station qui représente un point de passage obligatoire pour une grande partie du trafic comme un routeur ou une station de base. Nous développerons notre exemple sur une station de base.

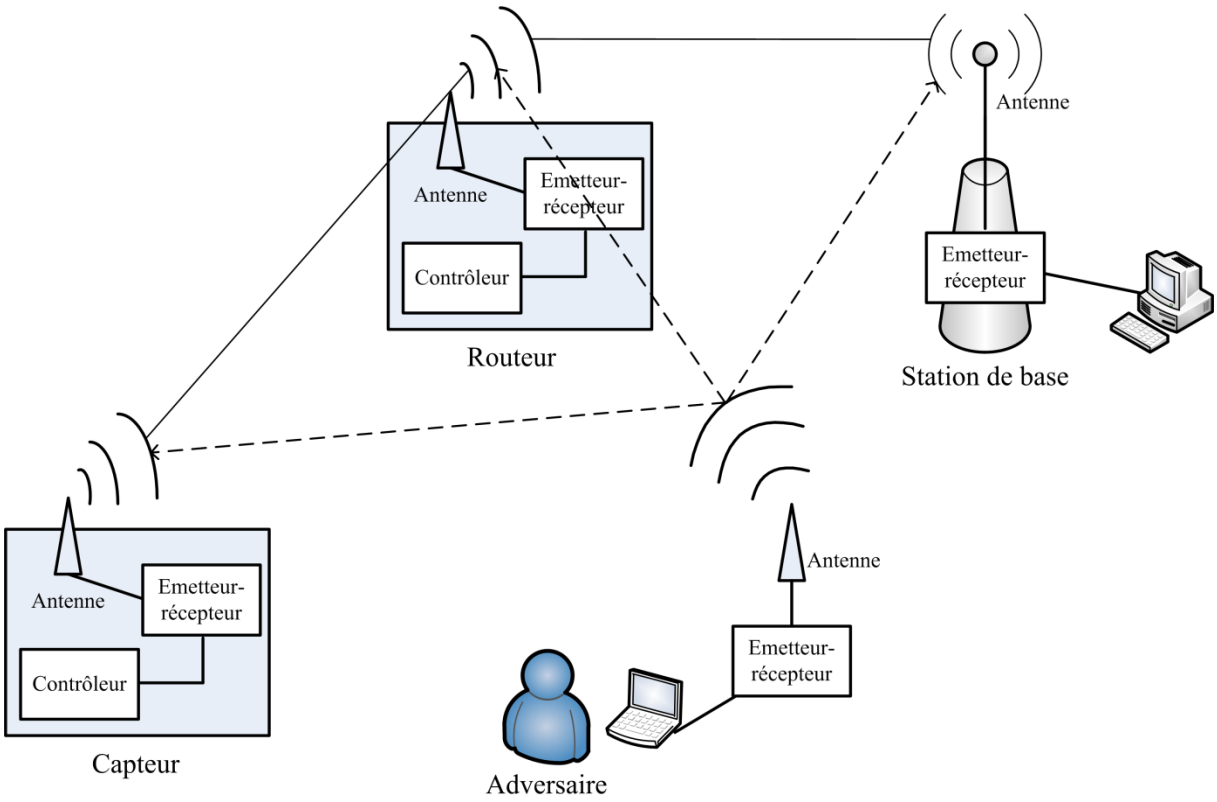


Figure II.2 : L'attaque par *Jamming*. Un adversaire envoie des signaux perturbateurs sur des fréquences de communication utilisées par les nœuds capteurs du réseau.

Il existe plusieurs techniques de défense contre le *Jamming* sur la station de base. Une première technique est la réplication de la station de base. En effet, la station de base remplaçante pourra prendre le relai si elle n'est pas attaquée elle aussi. La deuxième technique consiste à changer la place de la station de base, ce qui lui permet de s'éloigner de la source de l'attaque pour se trouver hors de portée du *Jamming*. La troisième technique est de changer le chemin des données (technique *multipath*). Dès qu'un nœud détecte un *Jamming* sur le chemin vers la station de base, il tente d'envoyer les données via un autre chemin. Dans [25], les auteurs proposent une technique hybride qui réunit les trois techniques présentées ci-dessus afin de se défendre des effets du *Jamming* sur une station de base.

Une technique qui peut être utilisée par tous les nœuds, est le saut de fréquences ou *frequency-hopping spread spectrum (FHSS)*. Le but est de permettre aux nœuds de changer de fréquence en suivant une séquence pseudo aléatoire connue par l'expéditeur et le destinataire (ou l'ensemble des destinataires) [26]. Afin d'attaquer le réseau dans ce cas, il faudra que l'adversaire arrive à occuper une partie importante des fréquences utilisées dans la séquence de sauts.

Notons qu'il existe des moyens simples qui permettent de limiter le déni-de-service causé par le *Jamming*. Par exemple, si un nœud reçoit en permanence des signaux et arrive à identifier que c'est un *Jamming*, il pourra alors les ignorer ou se mettre en mode veille pour un moment (ce qui provoquera une réaction au niveau du routage des paquets). Il se réveillera de temps en temps pour voir si l'attaque est toujours active [27].

II.1.2.2 Collision

Un nœud malveillant peut interférer dans une communication en transmettant un paquet « en sens inverse » pour causer une collision. En particulier, le nœud malveillant cible un paquet (ou trame) d'acquittement (ACK) afin de bloquer/retarder la communication ce qui, outre le débit, affecte la consommation d'énergie des nœuds. Pour mieux comprendre le déroulement de cette attaque, nous l'illustrons dans la Figure II.3. Un adversaire place un nœud malveillant M entre deux nœuds N1 et N2 qui communiquent entre eux. Il désire à travers ce nœud perturber leurs communications. Au début, l'adversaire écoute les échanges. Il remarque que le paquet 1 envoyé par N1 est suivi d'un acquittement ACK envoyé de la part du destinataire N2. Ainsi, l'adversaire mesure approximativement le temps nécessaire entre la réception du paquet et l'envoi de l'ACK de la part de N2. Il attend l'envoi du paquet 2 puis il

émet un paquet juste au bon moment pour provoquer une collision avec l'ACK. Après un certain temps d'attente, N1 va devoir renvoyer le paquet 2 car il n'a pas pu décoder l'acquittement et considère que le premier envoi du paquet 2 est infructueux. Cette attaque perturbe les échanges entre N1 et N2 et oblige ces nœuds à perdre leur énergie en renvoyant des paquets.

La détection d'une telle attaque est difficile car le temps d'attaque est court et les paquets envoyés par le nœud malveillant sont similaires à des paquets normaux.

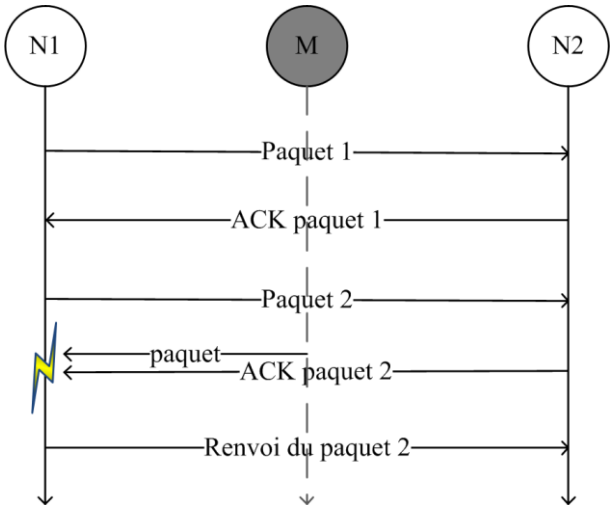


Figure II.3 : L'attaque par collision. Un nœud malveillant M se place entre deux nœuds pour essayer de perturber leurs communications en envoyant des paquets en même temps que les ACK.

II.1.2.3 *Sinkhole*

Cette attaque fait partie de l'ensemble des attaques qui causent un déni-de-service du réseau. Dans cette attaque, le nœud malveillant agit comme une station de base en attirant les paquets vers lui et en les empêchant de continuer leur chemin. L'attaque par *Sinkhole* est parfois assimilée à une attaque de type *Black Hole* (*i.e.*, le trou noir) où un nœud malveillant fabrique un trou noir au sens de la physique dans le réseau afin de détruire les paquets arrivant vers lui [28]. Cependant, l'attaque *Sinkhole* est plus complexe. Afin de préserver l'énergie des nœuds, les protocoles de routage dans les RCSF sont souvent basés sur des objectifs comme l'exploitation des chemins avec un minimum de sauts ou un minimum de délai sur les liaisons de bout-en-bout. Les attaques comme *Sinkhole* exploitent ces propriétés en proposant de faux chemins optimaux aux nœuds du réseau, c'est par ce fait qu'elles attirent les paquets vers un nœud qui va les absorber.

La Figure II.4 montre que l'attaque réussit à créer le trou noir afin d'absorber le trafic passant par le nœud malveillant M. Notons que M pourra envoyer des faux paquets à S à la place des paquets détruits si les communications ne sont pas chiffrées ni authentifiées.

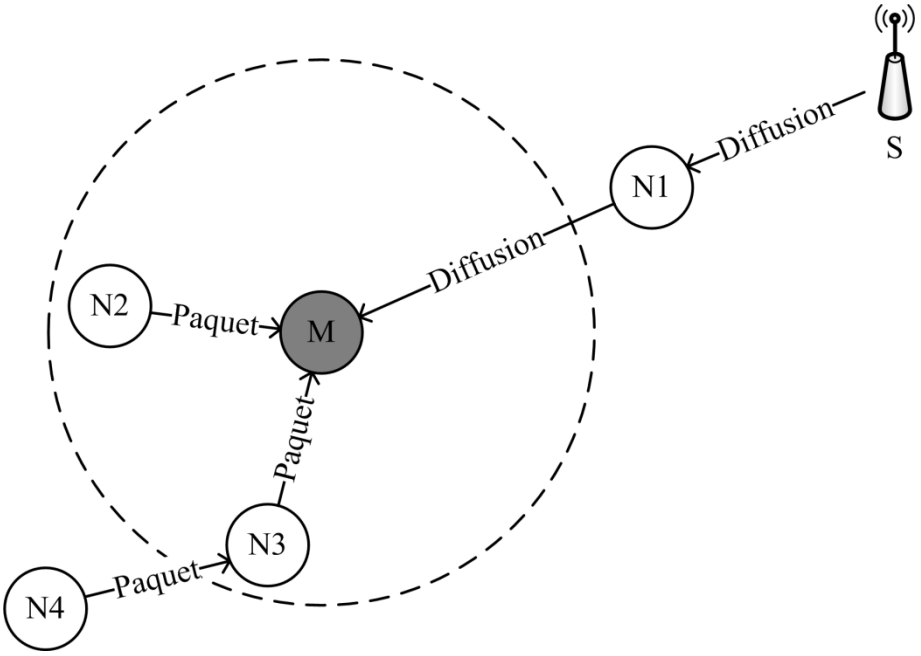


Figure II.4 : L'attaque par *Sinkhole*. Le nœud malveillant M attire des paquets pour les absorber. Il empêche par exemple la diffusion de proche en proche initiée par une station de base et destinée à ses fils.

Les protocoles de routage comme [29] et [30] appartiennent aux catégories des protocoles qui utilisent la position géographique des nœuds. Ils peuvent aider le réseau à résister contre de telles attaques. Quand les nœuds sont localisés, il est plus difficile pour un adversaire de proposer aux autres de meilleurs chemins sachant que la localisation des nœuds donne déjà une idée sur ce type d'information. D'autres solutions cryptographiques comme l'utilisation de la cryptographie à clé publique permettent de lutter contre ce genre d'attaques [18]. Ce point sera plus particulièrement traité dans la partie II.2.2.2, nous verrons que ces solutions utilisent généralement l'authentification des nœuds à l'aide de techniques asymétriques.

II.1.3 Agrégation de données et épuisement de ressources

Les attaques de ces deux catégories concernent les couches : liaison de données, réseau et transport.

II.1.3.1 *Sybil*

Un nœud malveillant peut prétendre avoir de multiples identités en utilisant les identités des nœuds ciblés par l'attaque. L'attaque *Sybil* [31] est le nom porté par ce genre de menaces. Les nœuds ciblés par l'attaque sont connus sous le nom de *Sybil nodes*. Cette attaque est localisée entre la couche liaison et la couche réseau. Elle vise à dégrader l'intégrité des données, le niveau de sécurité et l'utilisation des ressources (Figure II.5).

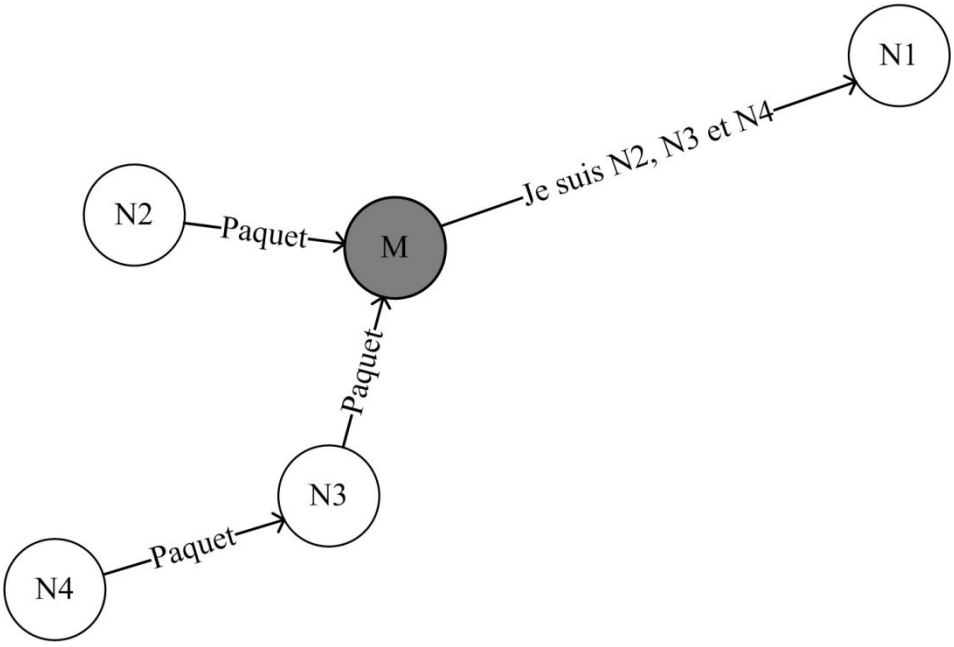


Figure II.5 : L'attaque *Sybil*. Un nœud malveillant M prend une place dans le réseau et reçoit les paquets de plusieurs nœuds, ici N2, N3 et N4. Il prétend être ces derniers auprès de N1 après avoir usurpé leur identité. Si N1 ne possède pas le moyen de vérifier l'identité des expéditeurs, M jouera le rôle des autres nœuds et l'attaque sera réussie.

L'utilisation de la cryptographie à clé publique constitue une solution contre cette attaque. Ceci consiste essentiellement à vérifier l'identité des nœuds voisins ou fils. Par contre, la génération et la vérification des signatures digitales sont faisables dans les RCSF mais coûteuses en termes de mémoire et temps de calcul. Dans [32], les auteurs proposent l'utilisation des certificats d'identité afin d'authentifier les nœuds, alors que les auteurs de [33] suggèrent que chaque nœud partage une clé symétrique unique avec une station de base (qui sera alors entité de confiance). Si un nœud N a besoin de communiquer avec un voisin V, il demande à la station de base de lui fournir les clés nécessaires pour établir un lien authentifié et chiffré entre N et V. D'autres mécanismes de détection et de prévention sont cités dans [34], [35] et [36].

II.1.3.2 *Man-in-the-middle*

L'attaque de l'homme du milieu ou *Man-in-the-middle* fait partie des attaques qui peuvent avoir lieu sur plusieurs couches comme l'attaque *Sybil.* Dans cette partie, nous allons utiliser deux personnes (Alice et Bob) à la place de deux nœuds du RCSF afin de mieux comprendre le déroulement de l'attaque. L'attaque de l'homme du milieu, comme l'indique son nom, est le fait qu'une personne étrangère au réseau (Ève) vienne prendre place au milieu entre Alice et Bob, intercepte leur conversation et empêche certains messages d'être délivrés à leurs destinations. Le but d'Ève est d'intercepter les messages de l'un, par exemple Alice, pour les transmettre le plus souvent modifiés à l'autre, c'est-à-dire Bob. Ève pourra atteindre son but sans qu'Alice et Bob s'aperçoivent que leurs échanges sont modifiés. Dans ce cas, le risque est grand, la plupart des services fondamentaux de sécurité sont brisés. Notons que le fait d'empêcher d'un message de continuer son chemin en RCSF peut être réalisé en envoyant au même moment un message à la destination pour causer une collision (Figure II.3). Cependant, dans certains cas, l'interception peut être difficile à réaliser, surtout si les nœuds sont mobiles ou si les liens sans fil sont par nature de caractéristiques évolutives.

Nous donnons dans la Figure II.6 un exemple d'une communication assurée à l'aide d'un système à clé publique. Dans cet exemple, Alice désire envoyer un message chiffré à Bob. Dans un premier temps, Alice demande de Bob d'envoyer sa clé publique. Lors de l'envoi de la clé publique de Bob, Ève peut intercepter le message (le paquet), l'empêcher de continuer son chemin et extraire la clé publique de Bob. Ève est en mesure à ce moment précis de jouer le rôle de Bob. Elle envoie sa propre clé publique à Alice en usurpant l'identité de Bob. Alice, de son côté, ne s'aperçoit pas du rôle joué par Ève et chiffre le message destiné à Bob avec la clé publique qu'elle a reçue. Une deuxième fois, Ève intercepte l'envoi et l'empêche de continuer son chemin. Mais cette fois du côté d'Alice. Elle récupère le message et le déchiffre avec sa clé privée. Elle peut modifier son contenu avant de le chiffrer à nouveau à l'aide de la clé publique de Bob. Ensuite, elle envoie le message chiffré à Bob. En recevant le message, Bob le déchiffre à l'aide de sa clé privée et récupère les données modifiées par Ève.

Notons que l'exemple de la Figure II.6 illustre la parfaite réussite de l'attaque. L'adversaire Ève n'a pas réussi seulement à casser la confidentialité de la communication par écoute, mais aussi à casser l'intégrité des données en modifiant le contenu des messages échangés.

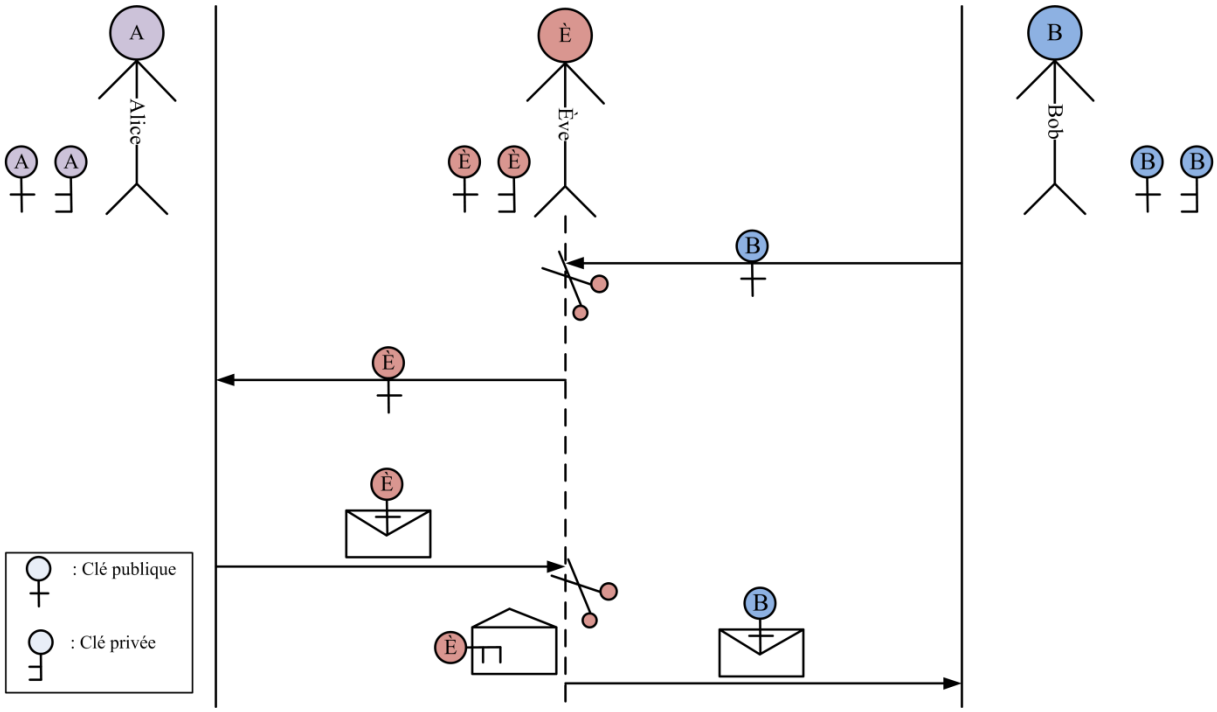


Figure II.6 : L'attaque de *Man-in-the-middle*. Alice désire envoyer un message à Bob. Ève s'étant placée au milieu, elle peut écouter et modifier les échanges. Elle joue le rôle de Bob auprès d'Alice et inversement.

Nous pouvons conclure de cet exemple que le vrai problème des systèmes à clé publique est le besoin de connaître la clé publique de son vrai correspondant pour pouvoir chiffrer les messages. Alice ne connaissait pas la clé publique de Bob et elle n'avait aucun moyen de vérifier que la clé reçue était vraiment celle que Bob avait envoyée [37].

Les solutions qui ont été proposées dans les réseaux LAN pour lutter contre ce genre d'attaques connues depuis l'origine des réseaux câblés sont basées sur l'utilisation des certificats numériques et l'infrastructure à clé publique PKI (*Public Key Infrastructure*). Le certificat fait le lien entre l'identité de la personne et sa clé publique. La PKI délivre et assure la signature des certificats. Cette infrastructure est détaillée dans la partie II.2.2.2.8. Plusieurs solutions spécifiques au RCSF ont été proposées dans la littérature afin de répondre à cette problématique (voir partie II.3). Elles sont basées sur une pré-distribution de clés avant le déploiement du réseau.

II.1.3.3 *De-Synchronization*

C'est une attaque qui fait partie des attaques d'épuisement de ressources. L'adversaire fait croire à un nœud qu'il n'a pas reçu un certain nombre de paquets en faussant la séquence des paquets réellement échangés. Pour cela, il suffit de changer le numéro de séquence des

paquets passant par lui. Il peut augmenter le numéro de séquence des paquets plus que prévu auprès de sa victime. En recevant les paquets modifiés, le destinataire en déduit que des paquets ont été perdus et réclame le renvoi de ces paquets auprès des expéditeurs. L'une des solutions est l'authentification de tous les paquets qui circulent dans le réseau [33].

II.1.4 Capture physique de nœuds

Une des caractéristiques des RCSF est le déploiement de leurs nœuds dans des environnements où ils sont censés être autonomes. Une fois déployés, les nœuds établissent les liens entre eux et constituent les chemins de routage sans intervention extérieure. Ces propriétés facilitent le rôle des adversaires surtout dans les régions d'accès facile où les nœuds peuvent être attrapés ou physiquement manipulés.

L'adversaire doit posséder les équipements techniques et la connaissance suffisante pour mettre en œuvre ce genre d'attaque. Il faut aussi prendre en considération les menaces liées à son intention comme le vol ou la destruction des nœuds capturés.

Nous pouvons classer les attaques *Tampering* et *Node Compromise* dans la même catégorie car elles ont besoin d'un accès physique aux nœuds. Nous détaillons ces deux attaques dans la suite.

II.1.4.1 *Tampering*

Après avoir capturé physiquement le nœud, l'adversaire cherche à l'exploiter pour déclencher une attaque de *Tampering*. Le but de cette attaque est de manipuler le nœud capturé en installant un nouveau code ou à travers l'extraction des données cryptographiques comme les clés de chiffrement. Ces données peuvent être utilisées ultérieurement pour déclencher d'autres types d'attaques.

Dans [38], les auteurs proposent une nouvelle méthode qui extrait les données de la mémoire d'un nœud capturé appelée « *JTAG attack* » (cela évoque une attaque via l'interface de programmation et de monitoring du nœud). Pour valider cette attaque, le test a été fait sur un nœud utilisant le protocole de sécurité TinySec [33]. Le but est de récupérer la clé secrète pré-distribuée et partagée entre ce nœud et le réseau. Après avoir fait quelques transformations dans la mémoire SRAM, il a été possible de localiser la clé secrète dans la mémoire et de la récupérer. D'autres travaux comme [39] et [40] suggèrent des schémas où

les clés pré-distribuées sont effacées après leur utilisation dans l'établissement des liens sûrs. Il faut alors s'assurer que le temps d'établir les liens et d'effacer les clés soit plus petit que celui de la capture physique et de l'extraction des données. Ceci montre le besoin de disposer d'un bon protocole de gestion de clés pour protéger les nœuds d'une telle attaque.

II.1.4.2 *Node Compromise*

Cette attaque est classée parmi l'une des plus dangereuses. L'adversaire se sert de moyens matériels et logiciels spécifiques pour réaliser l'attaque. Après la capture physique d'un nœud, l'extraction du code et des clés cryptographiques, l'adversaire pourra substituer à ce nœud un « dispositif remplaçant » plus puissant comme un ordinateur portable par exemple. Ce dernier permet de simuler l'architecture du nœud qu'il remplace et de le transformer en un nœud d'attaque en tant que « nœud compromis ». L'ordinateur portable aura bien sûr la même identité que le nœud qu'il remplace dans le réseau mais aussi de nombreux autres avantages comme une puissance de traitement plus importante, une plus grande taille mémoire, voire une plus grande puissance d'émission. L'attaque peut être effectuée maintenant de l'intérieur du réseau.

Un nœud compromis possède les propriétés suivantes :

- Le nœud reprogrammé exécute du code malveillant qui est différent du code exécuté sur un nœud légitime du réseau. Il cherche à voler les informations secrètes ou sensibles et/ou essaie de perturber le fonctionnement normal du réseau.
- Le dispositif remplaçant exploite une interface radio compatible avec les autres nœuds de telle sorte qu'il peut communiquer avec eux sans problème.
- Le dispositif remplaçant est un participant qui possède une identité dans le réseau. En supposant que la communication soit chiffrée et authentifiée par des primitives cryptographiques, l'appareil doit être en possession des clés secrètes d'un nœud légitime pour qu'il puisse participer aux communications chiffrées et authentifiées du réseau.

Il existe des moyens matériels pour éviter ce type d'attaques. Par exemple, lors de l'essai d'ouverture de la coque du nœud capteur, un mécanisme peut immédiatement effacer la mémoire ou déclencher une bombe (nœuds piégés utilisés par des applications militaires). Comme c'est généralement le cas quand nous voulons plus de sécurité, un tel matériel est

coûteux et imparfait. Il existe d'autres moyens moins sophistiqués et moins chers comme le camouflage ou l'enfouissement des nœuds. Ceci rend plus difficile l'accès physique aux nœuds.

L'authentification des nœuds peut protéger les réseaux contre les menaces provenant des nœuds extérieurs en refusant leur participation aux communications. Ceci ne résout pas le problème des nœuds compromis. Cependant, il existe une technique de détection d'intrusion qui permet d'identifier le fait qu'un nœud est compromis. La technique est connue sous le nom d'IDS (*Intrusion Detection Systems*) [41]. L'idée est de déployer avec les nœuds du RCSF des agents IDS afin de surveiller les échanges. La problématique de ces systèmes est la manière d'intégrer ces agents IDS dans les RCSF en respectant les limitations de ressources. Dans [42], les auteurs utilisent deux services importants d'un RSCF, l'agrégation et la localisation sécurisée, afin d'illustrer les efforts actuels de recherche concernant les IDS. Quand un nœud compromis est détecté, le réseau doit être capable de révoquer les clés cryptographiques utilisées par ce nœud.

II.2 La cryptographie dans les réseaux

Jusqu'à nos jours, les adversaires se basent sur l'absence d'une architecture de sécurité complète pour réaliser leurs attaques, c'est-à-dire, une architecture qui utilise tous les moyens disponibles et acceptables en coût pour protéger le réseau. Nous avons présenté dans la partie II.1 les différentes attaques qui menacent les réseaux en général et les RCSF en particulier. Néanmoins, il existe des attaques qui profitent des failles des systèmes de cryptographie pour déduire les données en clair ou bien pour trouver les clés utilisées, ces attaques font partie de la science de cryptanalyse. Dans un premier temps, nous commençons par introduire la science de la cryptanalyse puis nous enchaînons sur les systèmes cryptographiques utilisés dans nos jours.

II.2.1 La cryptanalyse

La cryptanalyse est la science de la reconstruction du texte en clair sans connaître la clé avec laquelle le texte a été chiffré. Le but est de fournir soit le texte en clair soit la clé.

II.2.1.1 Les attaques cryptographiques

Nous citons dans cette partie les attaques les plus connues sur la cryptographie. Le cryptanalyste essaie de récupérer le plus grand nombre d'informations sur le système de cryptographie utilisé ou les messages échangés afin de les exploiter dans une attaque parmi les suivantes :

- l'attaque à texte chiffré seulement : le cryptanalyste possède les textes chiffrés avec le même algorithme de plusieurs messages. Le Tableau II.1 contient, d'une part, les données dont le cryptanalyse a besoin et, d'autre part, les requis qu'il doit sortir à la fin de son analyse. Les données de son analyse sont constituées de messages chiffrés seulement. Pour que l'attaque réussisse, il doit être capable de fournir un algorithme permettant de décrypter le prochain message chiffré.
- L'attaque à texte en clair connu : le cryptanalyste a non seulement les textes chiffrés de plusieurs messages mais aussi les textes en clair correspondants. Le but est de retrouver la (ou les) clé(s) utilisée(s) pour chiffrer ces messages ou un algorithme qui permet de déchiffrer n'importe quel nouveau message chiffré avec la clé retrouvée.
- L'attaque à texte en clair choisi : cette attaque ressemble à l'attaque à texte en clair connu mais elle tire avantage du fait que le cryptanalyste est capable de choisir les textes en clair qui donneront le plus d'informations sur la clé de chiffrement.
- L'attaque à texte en clair choisi adaptative : c'est un cas particulier de l'attaque précédente. Non seulement le cryptanalyste peut choisir les textes en clair mais il peut également adapter ses choix en fonction des textes chiffrés précédents.
- L'attaque à texte chiffré choisi : le but est de retrouver la clé à l'aide d'un dispositif qui fait de déchiffrement.
- L'attaque à clé choisie : ce n'est pas une attaque où le cryptanalyste peut choisir la clé, il est seulement au courant de quelques relations entre différentes clés.

Dans le cas où le cryptanalyste ne dispose d'aucune information pour mettre en place les attaques citées précédemment, il pourra essayer toutes les combinaisons possibles pour retrouver la clé de chiffrement. Ceci est connu sous le nom « attaque par force brute ».

Données :	$\mathbf{C_1 = E_K(M_1)}$ $\mathbf{C_2 = E_K(M_2)}$ ... $\mathbf{C_i = E_K(M_i)}$	Où : E est la fonction de chiffrement, $\mathbf{M_i}$ désigne un message donné **i** ($\mathbf{i} \in \mathbb{N}$) et $\mathbf{C_i}$ est le message chiffré de $\mathbf{M_i}$ à l'aide d'une clé **K**.
Requis :	un algorithme permettant de déduire M_{i+1} à partir de C_{i+1}	

Tableau II.1 : Données et requis de l'attaque à texte chiffré seulement.

Données :	$\mathbf{M_1 et\ C_1 = E_{K_1}(M_1)}$ $\mathbf{M_2 et\ C_2 = E_{K_2}(M_2)}$ ... $\mathbf{M_i et\ C_i = E_{K_i}(M_i)}$	Où : E est la fonction de chiffrement, $\mathbf{M_i}$ désigne un message donné **i** ($\mathbf{i} \in \mathbb{N}$) et $\mathbf{C_i}$ est le message chiffré de $\mathbf{M_i}$ à l'aide d'une clé $\mathbf{K_i}$.
Requis :	- soit retrouver les clés K_i - soit un algorithme permettant de déchiffrer C_{i+1} en utilisant K_i	

Tableau II.2 : Données et requis de l'attaque à texte en clair connu.

II.2.1.2 Niveau de sécurité d'un algorithme

Le niveau de sécurité des algorithmes de chiffrement diffère selon différents critères [43]. Nous citons les critères les plus importants en les regroupant sous trois catégories :

- Algorithme **probablement sûr** : un algorithme est dit « probablement sûr »[1] s'il remplit l'une des conditions suivantes : (i) le coût nécessaire pour le casser dépasse la valeur de l'information chiffrée, (ii) le temps nécessaire pour le casser est plus long que la période durant laquelle l'information chiffrée doit rester secrète, (iii) les informations fournies par les textes chiffrés avec une même clé ne sont pas suffisantes pour le casser.
- Algorithme **inconditionnellement sûr** : un algorithme est dit inconditionnellement sûr si, quel que soit la quantité de textes chiffrés dont le cryptanalyse disposent, il n'y a pas d'information suffisante pour retrouver le texte en clair.

[1] On dit « probablement sûr » car il est toujours possible qu'une nouvelle avancée soit faite en cryptanalyse.

- Algorithme **invulnérable par calcul** : un algorithme est considéré invulnérable par calcul s'il ne peut pas être cassé avec les ressources disponibles actuellement et dans le futur.

Nous avons présenté dans la partie II.2.1 les diverses attaques sur la cryptographie puis le niveau de sécurité des algorithmes cryptographiques. Le choix d'un algorithme de chiffrement pour une application donnée reste un problème à résoudre avant toute proposition d'architecture de sécurité. Ce choix dépend en premier lieu du niveau de sécurité demandé. Par exemple, les besoins d'une application militaire visant à bien cacher ses communications secrètes diffèrent de ceux d'une application environnementale visant à garantir l'intégrité de ses communications.

II.2.2 Systèmes de cryptographie

Nous présentons dans cette partie les différents systèmes cryptographiques utilisés de nos jours afin de sécuriser les communications des réseaux. Nous commençons par le système de cryptographie symétrique puis nous détaillons le système de cryptographie asymétrique et ses primitives (échanges de clés, signature numérique, etc.), et enfin nous finissons cette partie par le système de cryptographie hybride qui réunit les deux systèmes symétrique et asymétrique.

II.2.2.1 Cryptographie symétrique

La cryptographie symétrique est la plus ancienne forme de chiffrement. Nous citons le vieil algorithme ROT13, un cas particulier du chiffre de *César* (prévu pour des textes écrits avec un alphabet de 26 caractères). L'idée est de faire un décalage de 13 caractères de chaque lettre du texte clair. Mais le problème de ce genre d'algorithmes, c'est que lui-même était secret. Il n'y avait pas de véritable notion de clé de chiffrement. Une fois le principe de l'algorithme découvert, nous pourrons déchiffrer tous les messages. Ensuite, les techniques modernes ont introduit un nouveau concept. Il s'agit de rendre les algorithmes publics et de leur associer la notion d'une clé secrète pour pouvoir chiffrer avec. Le chiffrement s'applique au niveau le plus bas, c'est-à-dire au niveau des bits constituant le message en utilisant une opération logique élémentaire (XOR ou OU exclusif) de ces bits et ceux de la clé. Nous distinguons deux types d'algorithmes : les algorithmes de chiffrement par blocs qui prennent

n bits en entrée et restituent n bits, et les algorithmes de chiffrement de flux *(stream algorithms)* qui travaillent bit par bit au fil de l'eau.

II.2.2.1.1 Principe

La cryptographie symétrique qui se base sur l'utilisation d'une seule clé de chiffrement est appelée « symétrique » ou « secrète », ceci pour signifier que la sécurité des entités communicantes repose sur le fait que cette clé ne soit connue que par les entités qui communiquent entre elles. Nous illustrons dans la Figure II.7 un exemple de chiffrement symétrique. Lorsqu'Alice désire envoyer un texte à Bob pour cela, elle partage avec lui, une clé secrète. Grâce à cette clé, Alice va pouvoir chiffrer le texte à envoyer à Bob. À son tour, Bob va pouvoir déchiffrer le texte en utilisant le même algorithme de chiffrement et la clé secrète.

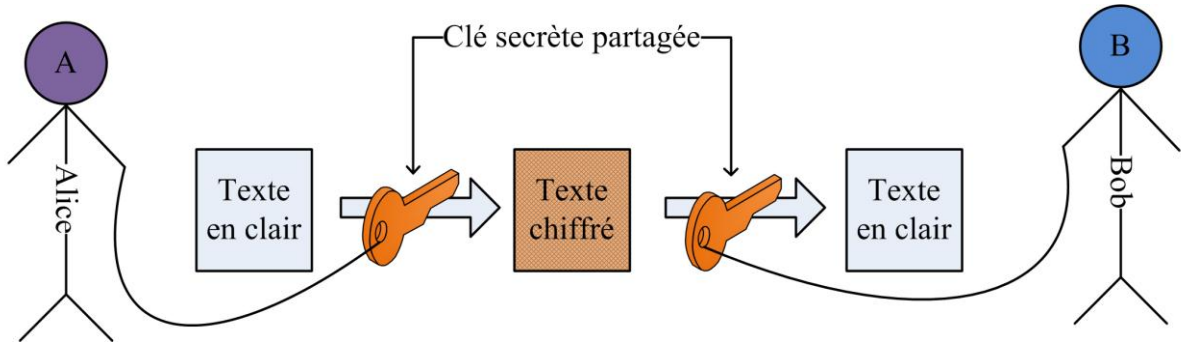


Figure II.7 : Chiffrement symétrique. Alice partage avec Bob une clé secrète afin de lui envoyer un texte chiffré.

II.2.2.1.2 Algorithmes de chiffrement symétrique

Nous présentons dans cette partie deux algorithmes de chiffrement symétrique. Le premier est l'algorithme standard AES *(Advanced Encryption Scheme),* très connu et très utilisé dans la cryptographie symétrique. Tandis que le deuxième, est l'algorithme basé sur la théorie de Chaos. Les chercheurs travaillent encore à présenter et à valoriser ses avantages. Il est beaucoup plus rapide qu'AES mais il reste encore à prouver qu'il puisse être aussi sûr et sécurisé.

II.2.2.1.2.1 L'algorithme AES

L'algorithme de chiffrement AES [44] est un algorithme symétrique approuvé par la NSA *(National Security Agency)* pour chiffrer les informations *top secret*. Il est connu aussi

sous le nom de *Rijndael*. Il a été conçu par *Joan Daemen* et *Vincent Rijmen*, deux chercheurs belges. Nous avons choisi de présenter cet algorithme parmi d'autres car il répond encore aujourd'hui aux exigences de sécurité internationale puisqu'il est encore considéré comme sûr même avec une taille de clé de 128 bits. Il n'a pas besoin de beaucoup de ressources et possède une flexibilité d'implémentation logicielle comme matérielle. Cela fait de lui un candidat particulièrement apprécié pour les implémentations embarquées où les ressources sont limitées.

C'est un algorithme de chiffrement par blocs, c'est-à-dire que le texte clair sera découpé en blocs avant d'être chiffré. Il supporte différentes combinaisons de couple (taille de clé, taille de bloc), nous citons (128, 128), (192, 128) et (256-128) bits. L'algorithme *Rijndeal* supporte des tailles de blocs différentes mais elles n'ont pas été retenues dans le standard. Le processus de chiffrement nécessite un certain nombre de tours de chiffrement (*rounds*). Le standard AES fixe 10 tours pour AES-128, 12 tours pour AES-192 et 14 tours pour AES-256. Le chiffrement commence par la fabrication d'une matrice appelée *State* contenant le texte clair et une fonction qui s'appelle *KeyExpansion* dédiée à la fabrication des parties dérivées de la clé de chiffrement. Le chiffrement s'applique sur la matrice *State* en répétant un ensemble d'opérations selon un nombre de tours fixé. Dans la Figure II.8, nous illustrons la phase de chiffrement et celle du déchiffrement de l'algorithme AES. Le chiffrement est composé de deux grandes phases. La première commence après l'ajout d'une première opération *d'AddRoundKey*. Elle se répète jusqu'à l'avant dernier tour. C'est ici que la deuxième phase est appliquée une seule fois pour finir le chiffrement. Elle exploite toutes les opérations de la première phase sauf l'opération *MixColumns.* Nous pourrons constater que la partie droite de la figure est la phase de déchiffrement et elle agit d'une façon inverse par rapport à la phase de chiffrement située à gauche.

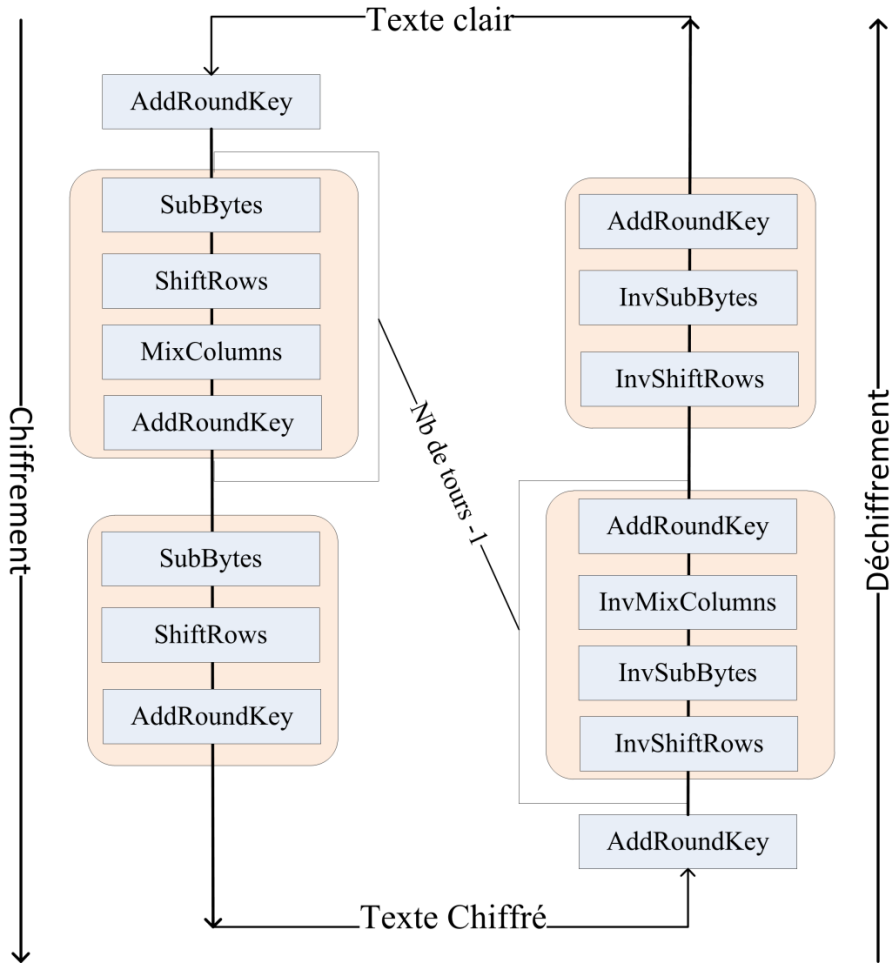


Figure II.8 : Structure de base de l'AES.

Voici l'ensemble des opérations basiques utilisées par l'AES :

- *AddRoundKey* : c'est la phase d'ajout d'une partie dérivée de la clé de chiffrement à la matrice *State.* L'ajout se fait en appliquant un XOR entre la partie dérivée et le texte clair de la matrice *State.*
- *SubBytes* et *InvSubBytes* : c'est la phase de substitution et son inverse. Nous substituons chaque octet par un octet résultant d'une transformation bien définie.
- *ShiftRows* et *InvShiftRows* : elles sont dédiées à changer l'ordre des octets dans chaque ligne. C'est un décalage d'un seul octet à chaque fois (sauf la première ligne).
- *MixColumns* et *InvShiftColumns* : elles utilisent une transformation qui s'applique à chaque colonne (4 octets) de la matrice *State.*

II.2.2.1.2.2 L'algorithme basé Chaos

Le chaos est un phénomène dans lequel une petite variation dans l'état initial du système provoque une divergence exponentielle dans l'évolution des états futurs. Cependant, il pourra perdre sa diversité et sa force lorsqu'il est utilisé dans un espace fini de nombres. L'utilisation de la précision finie est indispensable dans la cryptographie. Ceci affaiblit la longueur de cycles des orbites chaotiques qui tend vers l'infini.

Un algorithme basé sur la théorie de chaos a besoin de générer des valeurs de sortie afin de les utiliser pour le chiffrement. Le choix fait par de nombreux chercheurs de la carte PWLCM [45] *(Piece Wise Linear Chaotic Map)* est dû à ses nombreuses propriétés [46]. Nous citons parmi elles la densité invariante et la simplicité de son implémentation matérielle et logicielle. Les valeurs de sortie d'une carte PWLC sont présentées dans l'Équation II.1.

$$x(n) = F[x(n-1)] \quad = \begin{cases} x(n-1) \times \dfrac{1}{p} & si\ 0 \le x(n-1) < p \\ [x(n-1) - p] \times \dfrac{1}{0{,}5 - p} & si\ p \le x(n-1) < 0{,}5 \\ F[1 - x(n-1)] & si\ 0{,}5 \le x(n-1) \le 1 \end{cases}$$

Où p est une constante positive tel que $p \in]0, \frac{1}{2}[$ et $x(i) \in [0,1]$

Équation II.1 : Valeurs de sortie d'une carte PWLCM.

Même si la carte PWLCM assure de très bonnes propriétés, sa version numérique a des propriétés dynamiques très différentes de celles décrites par l'application continue, et certaines dégradations se présentent. Par conséquent, différentes solutions sont proposées pour améliorer la dynamicité de la dégradation numérique de PWLCM. Le problème dont souffrent les générateurs chaotiques numériques tels que les traditionnelles cartes chaotiques continues, c'est qu'ils sont discrétisés dans un espace fini 2^N qui conduit à de faibles propriétés statistiques. Comme les itérations chaotiques sont limitées dans un espace discret avec 2^N éléments, chaque orbite chaotique sera caractérisée par un comportement cyclique d'une durée limitée non supérieure à 2^N. Plusieurs solutions ont été proposées pour faire face à ce genre de problèmes : (i) utiliser la plus grande précision possible, (ii) perturber les sorties de la fonction chaotique, et (iii) utiliser plusieurs systèmes de chaos en cascade.

Pour une précision égale à N, chaque x peut être décrit comme :

$$x(n) = 0.x_1(n).x_2(n) \dots x_i(n) \dots x_N(n) \text{où } x_i(n) \in [0,1] \; et \; i = 1, 2, \dots, N$$

Dans la Figure II.9, nous présentons une perturbation à l'aide de la fonction LFSR *(Linear Feedback Shift Register)* appliquée à la sortie de la fonction de chaos. Une fois les sorties de la carte PWLC entrées dans un cycle périodique (répétition de valeurs de sortie), nous pourrons les obliger à sortir de ce cycle en appliquant une fonction de perturbation telle que LSFR.

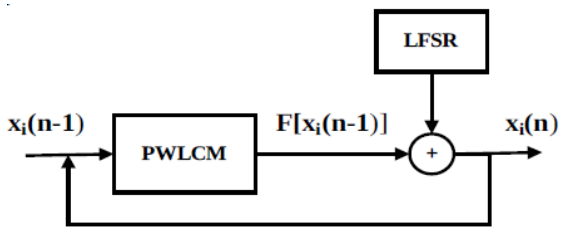


Figure II.9 : Perturbation d'une fonction chaotique [47].

L'Équation II.2 présente une génération de séquences de perturbation.

$$Q_{k-1}^{+}(n) = Q_k(n) = g_0 Q_0 \oplus g_1 Q_1(n) \oplus \dots \oplus g_{k-1} Q_{k-1}(n)$$

Où $g_0, g_1, \dots, g_{k-1}$ sont les coefficients du générateur polynomiale primitif et $Q_0, Q_1, \dots, Q_{k-1}$ sont les valeurs initiales du registre avec au moins une valeur non nulle.

Équation II.2 : Génération de séquences de perturbation.

La séquence chaotique présentée dans l'Équation II.3 est le résultat obtenu après l'application de la perturbation aux valeurs de sortie de PWLCM après chaque Δ itérations, où Δ est un entier positif qui désigne la longueur du cycle périodique.

$$x_i(n) = \begin{cases} F[x_i(n-1)] & si \; 1 \le i \le N-K \\ F[x_i(n-1)] \oplus Q_{N-i}(n) & si \; N-k+1 \le i < N \end{cases}$$

Équation II.3 : Séquence chaotique.

Dans [47], les auteurs proposent une méthode respectant les trois solutions présentées précédemment afin de renforcer la faiblesse causée par la dégradation de la carte PWLCM digitale. La méthode consiste à mixer et perturber deux sorties avant d'appliquer le chiffrement. Dans [46], les auteurs ont fait une analyse qui a abouti à un ensemble de conseils

et de précautions à prendre en considération lors de l'utilisation de ces trois solutions dans la cryptographie chaotique. Dans la Figure II.10, nous présentons le chiffrement et le déchiffrement de cette méthode. Après avoir partagé un certain nombre de paramètres (valeurs initiales), la source et la destination chiffrent et déchiffrent de la même manière. Et après avoir reçu un message chiffré C, le destinataire génère R_1et R_2 les deux valeurs perturbés. Il pourra obtenir le message initial en calculant l'Équation II.4.

$$M = C \oplus R_1 \oplus R_2 = (M \oplus R_1 \oplus R_2) \oplus R_1 \oplus R_2 = M \oplus (R_1 \oplus R_2 \oplus R_1 \oplus R_2)$$

Équation II.4 : Calcul du message initial.

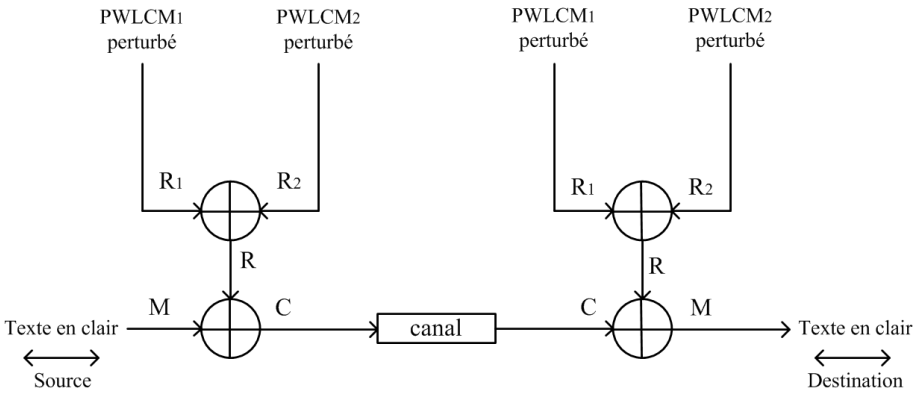


Figure II.10 : Une proposition basée sur deux sorties chaotiques perturbées régulièrement [47].

Notons que le partage de la clé symétrique est l'étape la plus compliquée dans une communication sécurisée par la cryptographie symétrique. Les deux entités communicantes doivent partager la clé secrète et la protéger chez elles durant toute la période de communication. Si cette clé est découverte par un adversaire, tous les échanges seront déchiffrés. Tant que la clé est à l'abri, le système symétrique est sûr. Cette problématique a poussé les chercheurs a inventé un système à paire de clés ou à clé publique.

II.2.2.2 Cryptographie asymétrique

La cryptographie asymétrique ou à clé publique s'oppose à la cryptographie symétrique du fait qu'elle utilise une paire de clés et non pas une seule. Elle permet non seulement le chiffrement d'informations mais aussi la garantie d'authentification des auteurs des messages en utilisant la signature numérique à l'aide d'un chiffrement par clé privée et l'utilisation des certificats numériques. Nous expliquons au début de cette partie le fonctionnement de la cryptographie asymétrique. Ensuite, nous présentons les méthodes de

chiffrement asymétrique les plus connues et leurs primitives ainsi que les infrastructures de gestion des certificats.

II.2.2.2.1 *Principe*

La cryptographie asymétrique ou « à clé publique » se base sur l'utilisation d'une paire de clés de chiffrement. Les clés asymétriques sont générées par paire. La clé publique est extraite et publiée à travers des annuaires, par exemple, tandis que la clé privée est gardée secrète chez l'utilisateur. Un message chiffré par la clé publique est déchiffré par la clé privée, et inversement.

Nous illustrons dans la Figure II.11 un exemple de chiffrement asymétrique. Lorsqu'Alice désire envoyer un texte à Bob, elle aura besoin de sa clé publique afin de chiffrer le texte. Bob va pouvoir déchiffrer, à l'aide de sa clé privée, le texte reçu en utilisant le même algorithme de chiffrement employé par Alice.

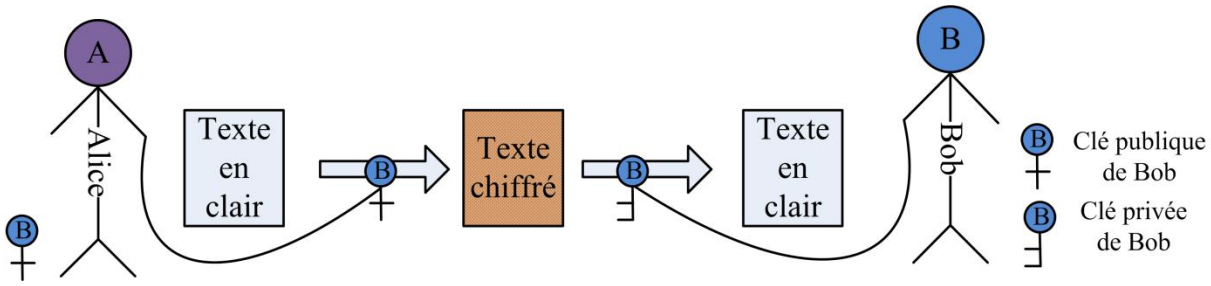


Figure II.11 : Chiffrement asymétrique. Alice utilise la clé publique de Bob afin de lui envoyer un texte chiffré.

Notons qu'Alice pourra obtenir la clé publique de Bob en la récupérant :

- ✓ de Bob. C'est au début de la conversation qu'Alice lui demande d'envoyer sa clé publique. Quand Bob répond avec sa clé publique non certifiée, la conversation pourra être interceptée par un tiers qui récupère la clé et la remplace par une autre. Cette menace est connue sous le nom d'attaque du *Man-in-the-middle* ou homme du milieu (voir partie II.1.3.2).
- ✓ d'un annuaire certifié. C'est une entité tierce de confiance. Elle contient des certificats générés par une autorité de certification. Pour éviter l'attaque de *Man-in-the-middle,* la clé publique de Bob est accompagnée de son identité et est certifiée (ou signée) par une autorité de certification. L'ensemble de ces

informations constitue ce qu'on appelle un certificat numérique (voir partie II.2.2.2.7).

Dans le premier cas, la sécurité de la communication entre Alice et Bob est menacée par une attaque très connue tandis que dans le deuxième cas, elle est assurée par l'obtention du certificat de Bob auprès d'une entité de confiance. La clé publique sera extraite du certificat et utilisée comme entrée d'un algorithme de chiffrement asymétrique.

II.2.2.2.2 *Diffie-Hellman*

La cryptographie à clé publique a été introduite pour la première fois par *Whitfield Diffie* et *Martin Hellman* de l'université de *Stanford*, en 1976 [48]. Pour résumer, les auteurs se sont basés sur des problèmes mathématiques et sur des fonctions difficilement inversibles (ou « à sens unique »). Ils ont démontré la possibilité d'utiliser un couple de clés au lieu d'une seule clé partagée. La clé publique est connue par tout le monde et la clé privée est gardée secrète chez l'utilisateur. Connaissant la clé publique, il est très difficile d'obtenir la clé privée. Il faut imaginer une fonction qui est à sens unique et qui est connue par tout le monde. La personne possédant la clé publique peut chiffrer à l'aide de cette fonction mais seule la personne possédant la clé privée est capable de faire l'opération inverse et déchiffrer les informations.

Diffie et *Hellman* se sont basés sur deux problèmes mathématiques pour pouvoir proposer leur protocole. Le premier est le DLP (*Discrete Logarithm Problem*) et le deuxième est le DHP (*Diffie-Hellman Problem*).

II.2.2.2.2.1 Le problème du logarithme discret

Le logarithme discret est une application réciproque à l'exponentiation. C'est l'analogue du logarithme qui est la réciproque de la fonction exponentielle. Nous présentons dans l'Équation II.5 le problème DLP.

Soient G un groupe fini d'ordre n et α un élément de G. Etant donné un élément $\beta \in G$, quelle est la valeur de x, $0 \leq x \leq n-1$, tel que $\alpha^x = \beta$?

Équation II.5 : Le problème DLP.

La valeur de *x* est considérée comme la clé privée et la valeur α^x est considérée comme la clé publique. Connaissant la clé publique, nous ne pouvons pas trouver facilement la clé privée.

II.2.2.2.2.2 Le problème de *Diffie-Hellman*

Le problème de *Diffie-Hellman* (DHP) est défini par le problème mathématique présent dans l'Équation II.6.

Etant donné un élément α et les valeurs α^x et α^y, quelle est la valeur de α^{xy} ?
(où x et y sont des entiers aléatoires et α est un générateur d'un groupe fini G).

Équation II.6 : Le problème DHP.

Il est (comme pour le problème du DLP) très difficile de calculer la valeur α^{xy} en connaissant seulement α, α^x et α^y. L'une des solutions serait d'essayer de trouver *x* et *y* mais ces valeurs sont gardées précieusement au sein des entités communicantes.

II.2.2.2.2.3 Le protocole d'échanges de clés

En se basant sur les deux problèmes présentés précédemment, *Diffie* et *Hellman* ont proposé un protocole sécurisé d'échange de clés qui permet d'établir une clé secrète entre deux entités utilisant un canal non sécurisé. La Figure II.12 montre comment le protocole *Diffie-Hellman* permet à Alice et Bob d'échanger des informations en clair afin de créer une clé secrète commune. Au début, Alice et Bob doivent se mettre d'accord sur un groupe fini cyclique *G* d'ordre *p* et un générateur *g* de *G*. Ces deux paramètres peuvent être envoyés en clair comme la figure le montre. Ensuite, Alice et Bob, chacun de leur côté, choisissent un nombre aléatoire (*a* et *b* respectivement) puis calculent les valeurs A et B (considérées comme leurs clés publiques). Après l'échange de ces valeurs, Alice et Bob sont capables de calculer le secret S. Le problème majeur des systèmes symétriques, le partage d'une clé, vient d'être résolu grâce à ce protocole.

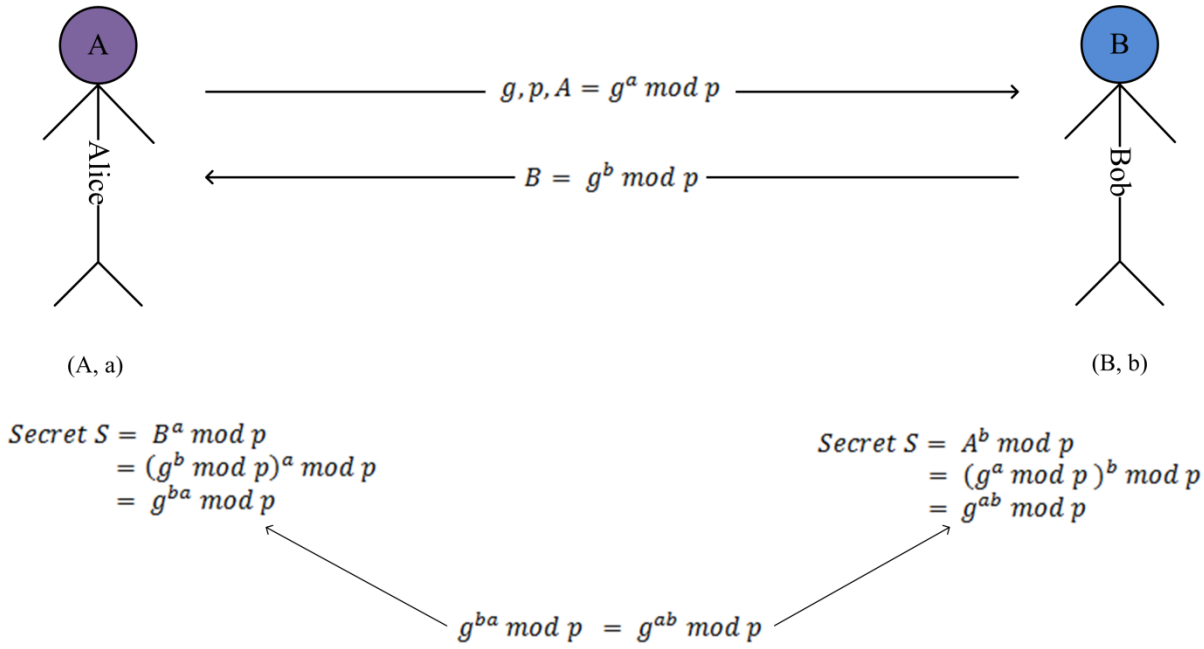


Figure II.12 : Le protocole *Diffie-Hellman*. Alice et Bob choisissent deux nombres aléatoires a et b, considérés comme clés privées. Alice calcule A, considérée comme clé publique, et l'envoie à Bob avec les paramètres g et p. Bob calcule B et l'envoie à Alice. Enfin, Alice et Bob calculent chacun de son côté, le secret S, considéré comme la clé secrète.

II.2.2.2.3 Chiffrement *El Gamal*

Le chiffrement *El Gamal* (ou *ElGamal encyption*) est une méthode de chiffrement à clé publique proposée par *Hatem El Gamal* en 1985 [49]. Elle est fondée sur le problème du logarithme discret et sur l'échange de clés de *Diffie-Hellman*.

Pour résumer la méthode de chiffrement d'*El Gamal*, nous considérons qu'Alice et Bob se sont mis d'accord sur un groupe fini G et g un générateur de G. Supposons qu'Alice désire chiffrer et envoyer un message $m \in G$ à Bob.

- Bob choisit d'abord un nombre premier p suffisamment grand pour que le calcul du logarithme discret soit infaisable. Ensuite, il tire un nombre aléatoire $k_B < p$ et calcule sa clé publique $B = g^{k_B} \bmod p$.
- Alice choisit un entier k au hasard et calcule $C_1 = g^k$ et $C_2 = m.B^k$ et envoie (C_1, C_2) à Bob.
- Bob retrouve $m = \frac{C_2}{{C_1}^{K_B}} = \frac{m.(g^{k_B})^k}{g^{k.k_B}}$.

Le défaut de cette méthode est que le message chiffré est deux fois plus long que le message en clair. Mais en terme de sécurité, casser le chiffrement revient à résoudre le DLP ce qui est difficile avec les moyens actuels.

II.2.2.2.4 La méthode RSA

La méthode de cryptographie RSA a été inventée en 1977 par *Ron Rivest, Adi Shamir* et *Len Adleman* [50], à la suite de la découverte de la cryptographie à clé publique par *Diffie* et *Hellman*. Dans les réseaux traditionnels, RSA est encore parmi les systèmes cryptographiques à clé publique les plus utilisés de nos jours. Il est basé sur le fait qu'il est facile de multiplier deux grands nombres premiers mais difficile de factoriser le produit [51]. Cette méthode est toujours considérée comme sûre, avec la technologie actuelle, pour des clés suffisamment grandes (1024, 2048, voire 4096).

Lorsque Bob souhaite recevoir des messages chiffrés provenant de la part d'Alice en utilisant la méthode RSA, il procède à la réalisation des quatre étapes suivantes :

1. **Génération de clés :**

   Bob crée quatre nombres *p*, *q*, *e* et *d* de la façon suivante :

   - p et q sont deux nombres premiers distincts. Leur génération se fait au hasard, en utilisant un algorithme de test de primalité probabiliste (voir Annexe VI.1.1).
   - soient $n = p \times q \; et \; \varphi(n) = (p-1)(q-1)$.
   - choisir *e* tel que *e* est premier avec $\varphi\,(n)$.
   - choisir *d*, tel que $e * d \equiv 1 \; mod \; (\varphi\,(n))$.

Pour que le chiffrement RSA soit sûr, il est nécessaire que le nombre $n = p \times q$ soit suffisamment grand pour que sa factorisation soit considérée comme très difficile à calculer. Il faut alors choisir d'abord de bons nombres premiers *p* et *q* (voir Annexe VI.1.3).

2. **Couple de clés**

   Le couple *(n, e)* constitue la clé publique de Bob et le couple *(n, d)* constitue sa clé privée.

3. **Envoi d'un message chiffré**

Alice transforme le message en un ou plusieurs entiers *M* compris entre *0* et *n-1*. Elle calcule dans ce cas, $C = M^e \ mod \ n$ qui sera envoyé à Bob.

4. Réception d'un message chiffré

Bob reçoit *C* et calcule grâce à sa clé privée $D = C^d \ mod \ n \ = \ (M^e)^d \ mod \ n = M \ mod \ n$, car $e * d \equiv 1 \ mod \ (\varphi \ (n))$. *D* est donc le message initial reconstitué.

Exemple :

- $p = 7$, $q = 19$
- $n = p \times q = 133 \ et \ \varphi(n) = (p-1)(q-1) = 108$
- $e = 5$ et $d = 65$
- soit $M = 14$, le message chiffré est alors : $C = 14^5 \ (mod \ 133) = 105$
- le message déchiffré est : $D = 105^{65} \ (mod \ 133) = 14$

II.2.2.2.5 Signature numérique

La signature numérique [52] a été créée dans le but d'authentifier l'expéditeur et de garantir l'intégrité des échanges dans les réseaux. Elle est basée sur les fonctions de hachage et le chiffrement avec la clé privée. Elle a été introduite après la découverte du système de chiffrement asymétrique. Nous présentons par la suite les fonctions de hachage, les étapes d'une signature d'un document et enfin nous donnons l'exemple de signature d'*El Gamal* proposée en 1985 et encore utilisée à ce jour.

II.2.2.2.5.1 Fonction de hachage

Une fonction de hachage produit un résumé (ou une empreinte) unique d'un texte. Le but de son utilisation est de rendre difficile la marche inverse, c'est-à-dire qu'un adversaire ne puisse pas obtenir le texte en partant de son empreinte comme le montre Figure II.13. Une bonne fonction de hachage permet d'obtenir un changement significatif de l'empreinte lors d'un petit changement du texte initial. Les algorithmes SHA-1 (*Secure Hash Algorithm 1*, 160 bits) [53] et MD5 (*Message-Digest algorithm 5*, 128 bits) [54, p. 5] sont les fonctions de hachage les plus utilisées. Cependant, MD5 n'est plus considéré comme sûr car il a été constaté qu'il pouvait y avoir « une collision » (deux messages différents ayant deux empreintes identiques) à partir de deux messages au contenu aléatoire [55] [56]. Ces deux

messages ont généré la même empreinte en utilisant MD5 ce qui n'est pas acceptable puisque l'empreinte est supposée être unique.

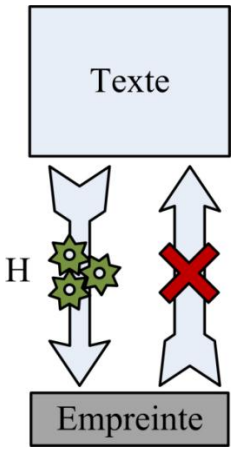


Figure II.13 : Fonction de hachage. Une fonction de hachage H fabrique une empreinte unique à un texte donné. Possédant cette empreinte, nous ne pouvons pas obtenir facilement le texte d'origine.

II.2.2.2.5.2 Signature d'un document

Notons que le verbe « signer » utilisé souvent en cryptographie veut dire « chiffrer à l'aide de la clé privée ». La signature numérique d'un document est réalisée en deux étapes. La première étape consiste à générer une empreinte à partir du document et la deuxième étape se charge de signer cette empreinte. Le document est envoyé au destinataire accompagné de sa signature. Le destinataire vérifie la signature du document afin d'authentifier l'entité expéditrice et de s'assurer de l'intégrité du document.

La Figure II.14 montre la génération de la signature du côté expéditeur et la vérification de la signature du côté destinataire. Le hachage a été utilisé pour deux raisons :

- ✓ signer le résumé du document est moins coûteux que signer tout le document,
- ✓ assurer l'intégrité des échanges.

Le moindre changement dans le document lors de son transfert conduit à un changement significatif dans l'empreinte *E*. Si la réponse de la comparaison de *E* avec *E'* est négative alors le document sera rejeté. Dans le cas où *E = E'*, la vérification est réussie et le document est accepté.

L'utilisation d'un système de chiffrement à clé publique est nécessaire pour la signature et pour la vérification. Pour signer le message, l'expéditeur utilise sa clé privée, et pour vérifier le message, le destinataire utilise la clé publique de l'expéditeur.

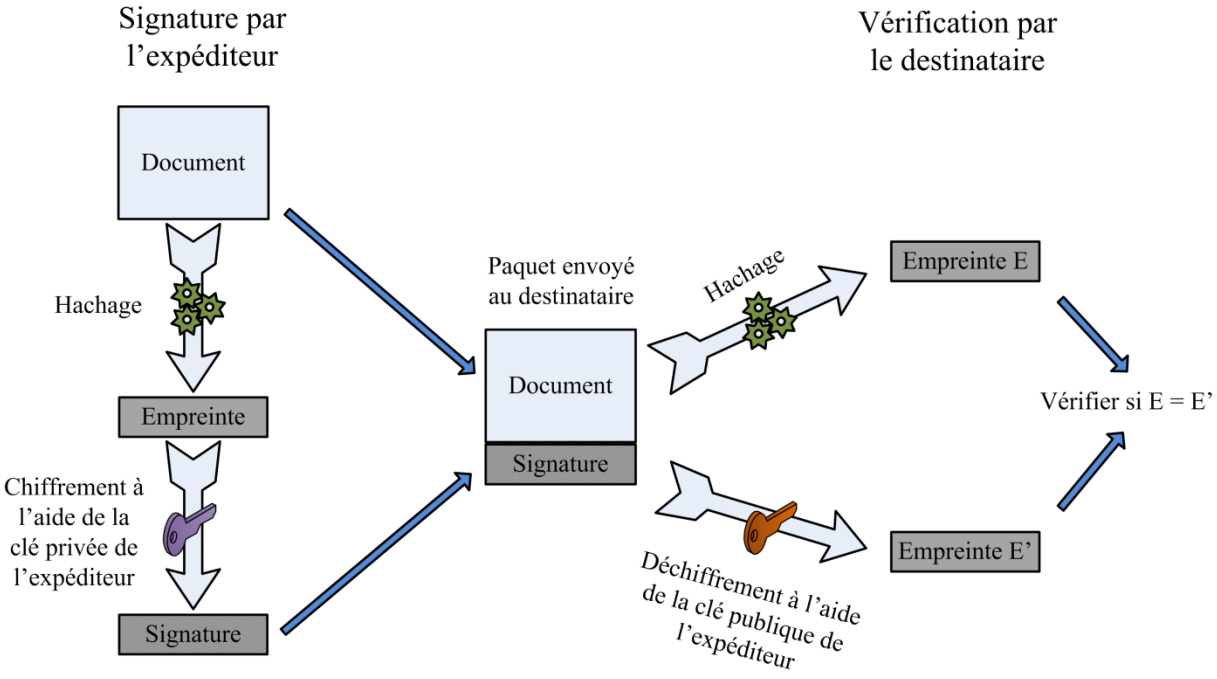


Figure II.14 : Signature numérique. Un document est résumé à l'aide d'une fonction de hachage puis chiffré avec la clé privée de l'expéditeur avant d'être envoyé. À la réception, le destinataire compare l'empreinte du document à celle obtenue après le déchiffrement.

II.2.2.2.5.3 Signature *El Gamal*

Nous présentons dans cette partie la méthode de signature d'*El Gamal* [49]. La méthode consiste à fabriquer un couple *(r, s)* envoyé comme signature au destinataire. Soient *G* un groupe cyclique, *g* un générateur de *G*, *p* un grand nombre premier, *h* une fonction de hachage et (P_A, k_A) la paire de clés d'Alice (P : clé publique, k : clé privée). Pour signer un entier m, $0 \leq m < p-1$, nous choisissons au hasard un entier k dans le même intervalle de m ($0 \leq k < p-1$) à condition qu'il soit premier avec $p-1$ et nous calculons le couple *(r, s)* de l'Équation II.7.

$$r = g^k \bmod p$$

$$s = (h(m) - k_A r)k^{-1} \bmod p - 1$$

Équation II.7 : calcul du couple de signature *(r,s)*.

Le couple *(r, s)* obtenu est la signature du message *m*. Il sera envoyé à Bob afin qu'il vérifie l'authenticité de sa signature. Pour ce faire, Bob doit disposer de la clé publique d'Alice P_A. Il vérifie si $0 \leq r \leq p$, $0 \leq s \leq p$ et $g^{h(m)} = {P_A}^r \, r^s \, mod \, p$.

II.2.2.2.6 Les courbes elliptiques

A l'origine, les courbes elliptiques ont été créées afin de résoudre des problèmes mathématiques fondamentaux sans avoir la moindre arrière-pensée sur le fait qu'elles auraient un grand succès en cryptographie. Elles ont été proposées indépendamment par *Victor Miller* [57] et *Neal Koblitz* [58] en 1985 pour être utilisées dans la cryptographie. Leurs applications sont aujourd'hui au centre de la cryptographie moderne et remplacent de plus en plus les grands standards. Elles ont détrôné RSA de sa position dominante. En effet, les algorithmes basés sur les courbes elliptiques sont plus rapides et aussi sûrs avec des clés plus petites.

II.2.2.2.6.1 ECC

Nous présentons dans cette partie la cryptographie basée sur les courbes elliptiques connue sous le nom d'ECC *(elliptic curve cryptography)*.

Soient F_q un corps fini à q éléments et $\overline{F_q}$ la clôture algébrique de F_q, c'est-à-dire que tout polynôme de degré supérieur ou égal à 1, à coefficients dans $\overline{F_q}$, admet au moins une racine dans $\overline{F_q}$. Une courbe elliptique E est l'ensemble des couples ou points $(x, y) \in \overline{F_q} \times \overline{F_q}$ vérifiant l'Équation II.8.

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6, \text{ avec } a_i \in F_q.$$

Équation II.8 : Equation générale de courbes elliptiques.

L'Équation II.8 est connue sous le nom de l'équation de Weierstrass sur le corps F_q. Pour leur usage en cryptographie, a_1, a_2 et a_3 doivent avoir une valeur nulle. Avec les cryptographes a_4 est devenu a et a_6 est devenu b, d'où l'Équation II.9 assortie d'une condition portant sur les coefficients.

$$y^2 = x^3 + ax + b, avec \; 4a^3 + 27b^2 \neq 0$$

Équation II.9 : Equation d'une courbe elliptique utilisée dans la cryptographie.

La courbe elliptique utilisée dans la cryptographie est constituée d'un ensemble de points *(x, y)* avec un point particulier nommé le point à l'infini $\mathcal{O}$, $E = \{(x, y) \in \overline{F_q} \times \overline{F_q}\} \cup \{\mathcal{O}\}$. L'ensemble des points de *(E)* forme un groupe abélien (groupe dont la loi de composition interne est commutative) par rapport à la loi d'addition, qui est une loi spécifique pour les courbes elliptiques, expliquée notamment dans [59]. Le point $\mathcal{O}$ va servir comme l'élément d'identité ou d'élément neutre de *(E)*. Notons que souvent le corps F_q est choisi d'une façon telle que $q = p^m$, p étant un nombre premier ($p = 2$, typiquement) et m étant la taille de la clé de chiffrement ($m = 160$, typiquement).

La Figure II.15 montre l'addition de deux points sur une courbe elliptique de la forme *(E)*. Pour additionner les deux points P_1 et P_2, nous traçons la droite qui passe par ces deux points. La droite coupe la courbe en un point P_3. Puisque les trois points sont alignés alors nous obtenons $P_1 + P_2 + P_3 = \mathcal{O}$. Alors, $P_1 + P_2 = -P_3$. Il reste donc à calculer le point P_4, l'opposé de P_3, pour obtenir l'addition. Dans le cas où $P_1 = P_2$ la droite qui passe par P_1 et P_2 est bien la tangente à P_1. Une autre opération importante est la multiplication scalaire. Par exemple, pour calculer $2.P_1$ nous faisons l'addition $P_1 + P_1$. Soit k un entier positif, le calcul de $k.P_1$ est égale à $\underbrace{P_1 + \cdots + P_1}_{k}$. La constante k est considérée comme une clé privée et le point obtenu sera considéré comme la clé publique après avoir vérifié qu'il appartient à la courbe. Pour plus de détails sur les opérations d'addition et de multiplication d'ECC avec des exemples, veuillez-vous référer à *Koblitz et al.* [59].

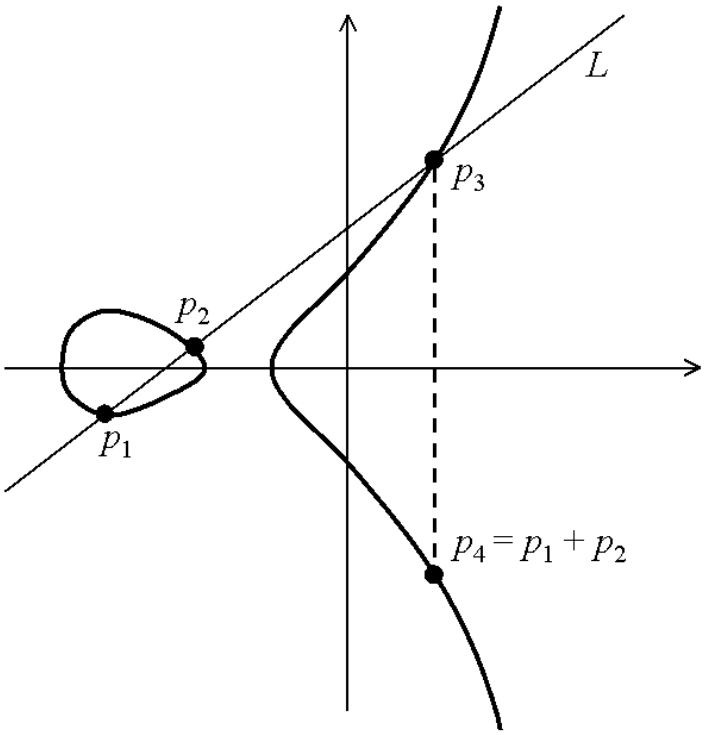


Figure II.15 : Addition de deux points sur une courbe elliptique. (source : http://www.rsa.com)

II.2.2.2.6.2 ECDH

ECDH (*Elliptic Curve Diffie-Hellman)* est l'adaptation de la méthode d'échanges de clés de *Diffie-Hellman* en se basant sur les courbes elliptiques. Alice et Bob doivent se mettre d'accord sur le choix de la courbe elliptique et fixer certains paramètres. Supposons que E est la courbe elliptique sur un corps fini F_q et Q est un point sur la courbe. Alice choisit secrètement un entier k_A et calcule le point $k_A.Q$ qui sera envoyé à Bob. A son tour, Bob choisit secrètement k_B et calcule le point $k_B.Q$ qui sera envoyé à Alice. Alice multiplie le point reçu par k_A et Bob le multiplie par k_B. Le secret partagé calculé des deux côtés est le point $k_A.k_B.Q$. Un adversaire connaissant Q, $k_A.Q$ et $k_B.Q$ mais pas k_A ni k_B doit résoudre le problème du logarithme discret DLP pour calculer le secret partagé.

II.2.2.2.6.3 ECIES

ECIES *(Elliptic Curve Integrated Encryption Scheme)* est un algorithme de chiffrement à clé publique basé sur les courbes elliptiques qui est une variante de celui d'*El Gamal* standardisée. La méthode de chiffrement est conçue pour être sémantiquement sécurisée en présence d'un adversaire capable de lancer des attaques comme les attaques à

texte en clair choisi et les attaques à texte chiffré choisi. Pour plus de détails sur cet algorithme, veuillez-vous référer à la partie 5.1 de [60].

Par la suite, nous présentons un exemple d'échange d'un message en utilisant ECIES. Soient *E* une courbe elliptique d'ordre premier *p* et *Q* un point appartenant à la courbe *E*. Nous définissons quelques notations :

- SYM est un algorithme de chiffrement symétrique (AES par exemple)
- MAC (code) un code authentificateur de message (*Message Authentication Code)*
- KDF une fonction de génération de clés *(Key Derivation Function)*

Supposons qu'Alice désire chiffrer et envoyer un message *m* à Bob. La clé privée de Bob est k_B et sa clé publique est $k_B.Q$. Nous présentons dans le Tableau II.3 les étapes qu'Alice et Bob doivent suivre afin de chiffrer et déchiffrer *m*. Notons que *x(Z)* dans la suite désigne l'abscisse du point *Z*.

Chiffrement côté Alice	**Déchiffrement côté Bob**
1. Choisir $\boldsymbol{k} \in [\mathbf{1}, \boldsymbol{p} - \mathbf{1}]$ au hasard	1. Vérifier que $R \in E$
2. Calculer $\boldsymbol{R} = \boldsymbol{kQ}$ et $\boldsymbol{Z} = \boldsymbol{k}.\boldsymbol{k_B}$	2. Calculer $Z = k_B R$
3. Calculer $\boldsymbol{k_1}\|\|\boldsymbol{k_2} = \boldsymbol{KDF}(\boldsymbol{x}(\boldsymbol{Z}), \boldsymbol{R})$	3. Calculer $k_1\|\|k_2 = KDF(x(Z), R)$
4. Calculer $\boldsymbol{C} = \boldsymbol{SYM}(\boldsymbol{k_1}, \boldsymbol{m})$	4. Calculer $t = MAC(k_2, C)$
5. Calculer $\boldsymbol{t} = \boldsymbol{MAC}(\boldsymbol{k_2}, \boldsymbol{C})$	5. Rejeter si $t \neq t'$ (reçu d'Alice)
6. Envoyer $(\boldsymbol{R}, \boldsymbol{C}, \boldsymbol{t})$ à Bob	6. Retrouver $m = SYM^{-1}(k_1, C)$

Tableau II.3 : Les étapes de l'algorithme ECIES.

Notons que cet algorithme n'est rien que la méthode de *Diffie-Hellman* avec un chiffrement symétrique et un mécanisme qui va servir à authentifier le message *m*.

II.2.2.2.6.4 ECDSA

ECDSA *(Elliptic Curve Digital Signature Algorithm)* est un algorithme de signature numérique à clé publique. La méthode de signature de cet algorithme est une variante de l'algorithme de signature DSA (*Digital Signature Algorithm*), proposée par le NIST *(National Institute of Standards and Technology)* en 1991 pour être utilisée dans son standard de signature DSS *(Digital Signature Standard)* [61]. Elle est basée sur les courbes elliptiques et varie très peu de la méthode de signature *El Gamal*.

Soient *E* une courbe elliptique d'ordre premier *p*, *Q* un point appartenant à *E* et *H* une fonction de hachage dans $[1,\ p-1]$. Supposons qu'Alice doive signer et envoyer un message $m \in [0,\ p-1]$. Afin de signer ce message, Alice doit d'abord créer une clé privée k_A et une clé publique $P_A = k_A.Q$. Bob doit disposer de la clé publique d'Alice pour pouvoir vérifier l'authenticité de la signature. Alice et Bob procèdent ensuite aux étapes présentes dans le Tableau II.4.

Signature côté Alice	Vérification côté Bob
1. **Calculer** e = H(m) 2. **Choisir** $\boldsymbol{k} \in [\boldsymbol{1}, \boldsymbol{p}]$ au hasard 3. **Calculer** $\boldsymbol{kQ}$ 4. **Convertir** $\boldsymbol{x}(\boldsymbol{kQ})$ en un entier $\bar{\boldsymbol{x}}$ 5. **Calculer** $\boldsymbol{r = \bar{x}\ mod\ p}$ 6. **Calculer** $\boldsymbol{s = k^{-1}(e + k_A r)\ mod\ p}$ 7. **Envoyer** (m, r, s)	1. **Calculer** e = H(m) 2. **Vérifier** que $r, s \in [1, p-1]$ 3. **Calculer** $w = s^{-1}\ mod\ p$ 4. **Calculer** $t_1 = ew, t_2 = rw\ mod\ p$ 5. **Calculer** $X = t_1 Q + t_2 P_A$ 6. **Convertir** $x(X)$ en un entier $\bar{x}$ 7. **Accepter si** $\bar{x} = r\ mod\ p$

Tableau II.4 : Les étapes de l'algorithme ECDSA.

II.2.2.2.7 Certificats numériques

Les clés asymétriques sont générées par paire : la clé privée est stockée et protégée chez l'utilisateur et la clé publique est ajoutée à un ensemble d'informations utiles à l'utilisateur pour constituer un certificat.

II.2.2.2.7.1 Le certificat d'identité X.509

X.509 est une norme de cryptographie de l'UIT[2] (Union Internationale des Télécommunications) pour les infrastructures à clés publiques *(Public Key Infrastructure, PKI)*. Le certificat d'identité X.509 *(Public Key Certificate, PKC)* est le document émis et signé par une autorité de confiance dite CA *(Certification Authority)*, associant une clé publique à des informations relatives au propriétaire du certificat. Il consiste à lier l'identité d'une entité à sa clé publique. Les informations du certificat sont examinées et vérifiées par la CA avant que le certificat ne soit validé. Ensuite, la CA signe le certificat après l'avoir condensé (c'est-à-dire, appliquer une fonction de hachage au certificat pour avoir un résumé)

[2] L'UIT est l'institution spécialisée des Nations Unies pour les technologies de l'information et de la communication

grâce à sa propre clé privée. Il suffit donc de connaître la clé publique de la CA, dont le certificat est largement distribué, pour vérifier la validité des certificats qu'elle génère.

Le certificat d'identité X.509 est considéré comme un format standard de certificats électroniques et un algorithme pour la validation de la chaîne de certification. Aujourd'hui, le format X.509 est utilisé par la plupart des protocoles de sécurité normalisés et mondiaux tels que SSL *(Secure Sockets Layer)*, TLS, IPSec, etc. Nous l'illustrons dans la Figure II.16. Un certificat X.509 contient des informations diverses telles que les noms et les identifiants du propriétaire et de la CA, la clé publique du propriétaire, la période de validité, et la signature de la CA. Le but est de créer une liaison entre la clé publique d'une entité et son identité à travers une signature d'une CA de confiance.

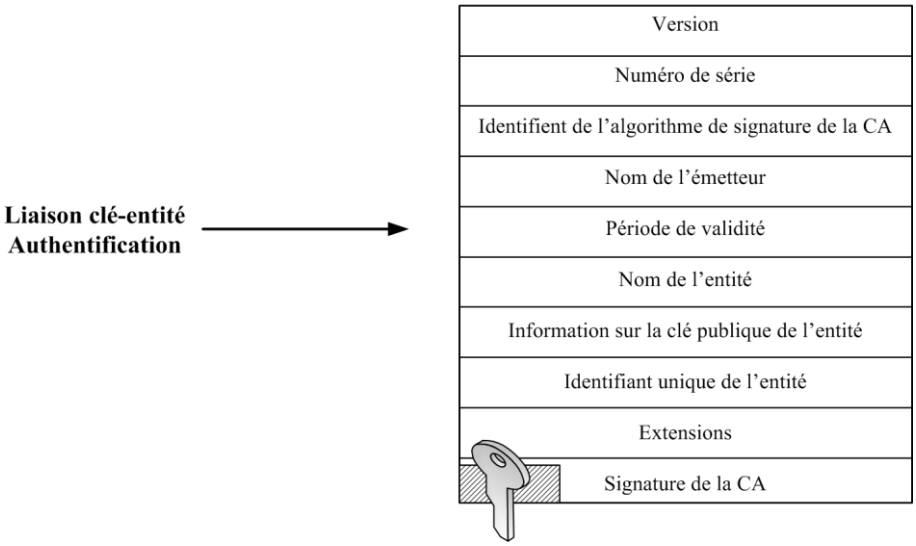


Figure II.16 : Certificat d'identité au format X.509.

Les certificats de deux entités peuvent être signés par deux autorités de certification différentes. Dans ce cas, il faut chercher une CA commune pour que les deux entités se fassent confiance. La Figure II.17 montre la chaîne de vérification construite par Alice et Bob. Pour que Alice vérifie le certificat de Bob, il lui faut d'abord obtenir les certificats des CA auprès des annuaires, puis extraire les clés publiques des certificats, et enfin terminer par la vérification des signatures jusqu'à l'arrivée à une CA commune. Une fois la chaîne construite, Alice pourra être sûre que la clé publique de Bob est vraiment celle extraite du certificat.

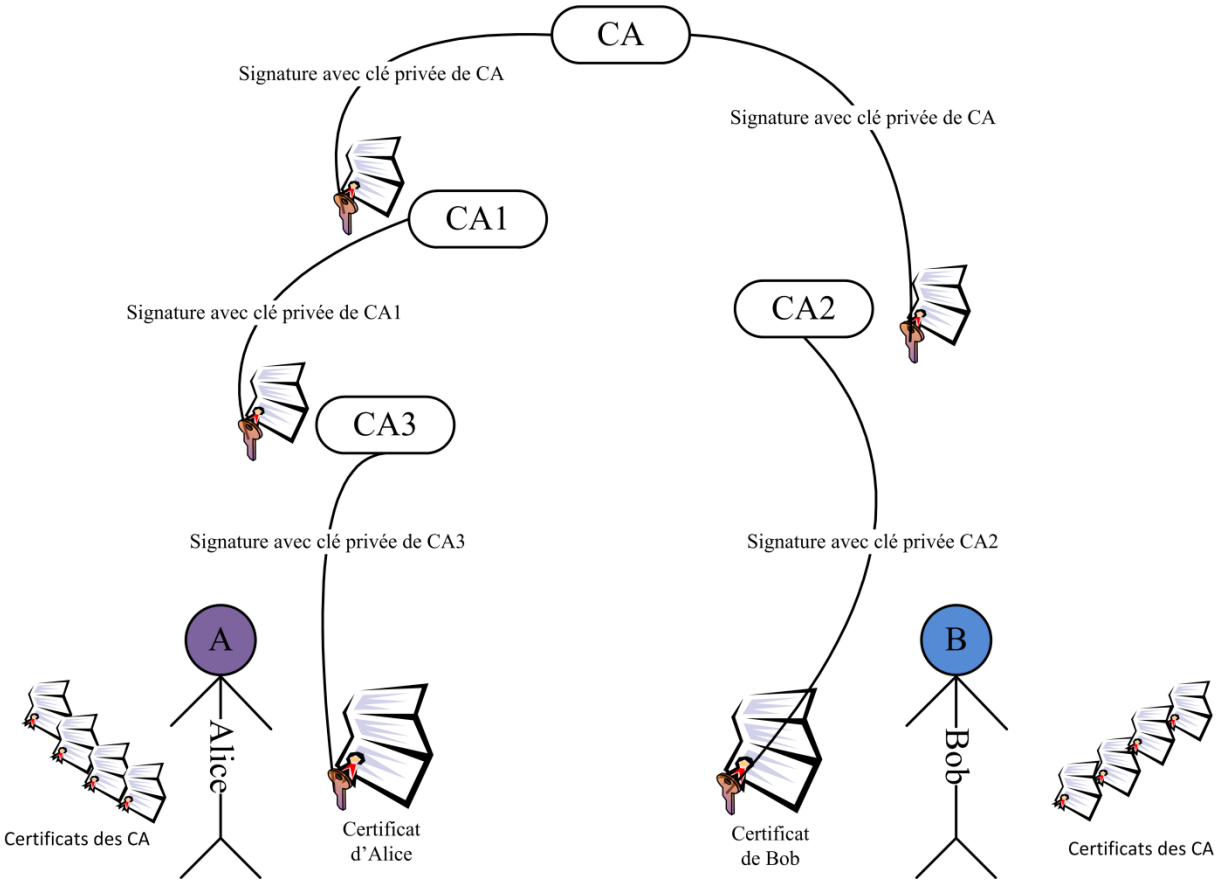


Figure II.17 : Chaîne de vérification de certificats. Le certificat d'Alice est certifié par CA3 et celui de Bob par CA2. Alice et Bob remontent une chaîne pour atteindre une autorité de certification commune.

Les certificats d'identité PKC ont été créés pour répondre aux besoins tels que : (i) l'authentification des entités, (ii) l'extraction d'une clé publique valide et certifiée pour l'utiliser dans le chiffrement asymétrique, et (iii) surtout pour se défendre contre les attaques comme l'homme du milieu (voir partie II.1.3.2). Pourtant, il ne répond pas à tous les besoins. En effet, le certificat, une fois signé, ne peut pas être modifié et par suite :

- la durée de vie des attributs n'est pas forcement la même que celle du certificat,
- les attributs peuvent être donnés par une entité différente de l'autorité de certification émettrice du certificat,
- les attributs ne sont pas nécessairement demandés au même moment que la demande du certificat.

Nous pouvons conclure que nous avons besoin des certificats spécifiques pour répondre à des besoins précis comme les permissions ou les autorisations.

II.2.2.2.7.2 Le certificat d'attribut

Le certificat d'attribut AC *(Attribute Certificate)* est un certificat similaire au PKC mais la différence est qu'il ne contient pas une clé publique [62].

Une analogie est souvent faite entre le PKC et l'AC d'une part et le passeport et le visa d'une autre part. Le PKC est utilisé comme un passeport. En effet, le but de ce dernier est d'identifier la personne ou l'entité. Il porte donc des informations sur son identité. Alors que l'AC est considéré comme un visa d'entrée. En effet, le visa n'a pas une longue durée de vie et il est généré par différentes autorités. Comme, pour obtenir le visa, nous devons présenter notre passeport, nous devons être identifiés par notre PKC avant que nous obtenions l'AC.

Les informations d'autorisation et les attributs sont mis dans l'AC et non dans le PKC pour deux raisons :

- la durée de vie des attributs n'est pas forcement la même que celle du PKC,
- les attributs peuvent être donnés par une entité différente de l'autorité de certification émettrice du PKC.

L'AC permet de donner la permission d'accès à une ressource, ce qui n'est pas le cas du PKC. Lors de l'accès à une ressource, il est parfois nécessaire de s'assurer que l'entité qui lance la requête et l'entité porteuse de l'AC sont identiques. La création d'une référence au PKC dans l'AC devient alors nécessaire et la clé privée correspondant au PKC est utilisée pour l'authentification lors de la requête d'accès.

L'AC pourra être aussi utilisé pour l'authentification des origines des données et la non-répudiation. L'AC contient les attributs et la signature de l'autorité de confiance. Cette signature va nous permettre de vérifier si le porteur de l'AC a le droit de signer des données.

Nous présentons dans la Figure II.18 l'AC avec ses attributs. Le but est de faire une liaison entre l'entité et ses permissions. L'AC est plutôt utilisé dans les réseaux distribués où les entités doivent posséder des permissions spécifiques pour qu'elles puissent exécuter des tâches à distance.

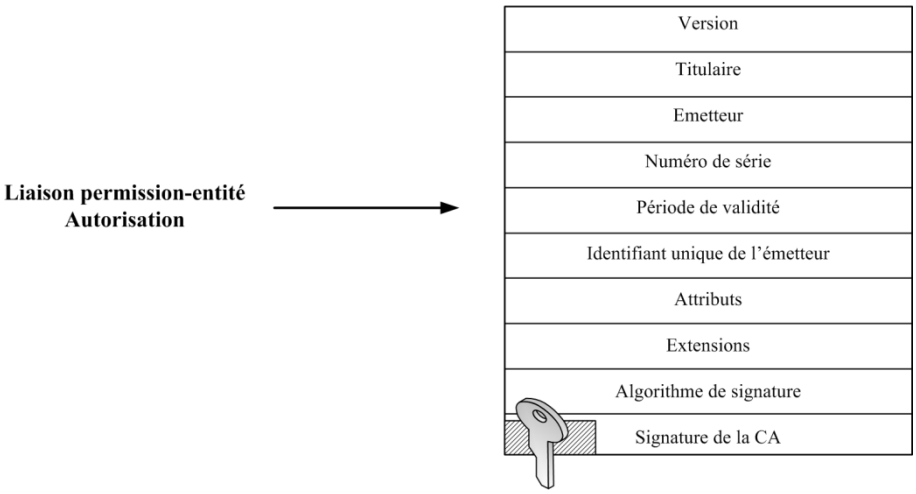


Figure II.18 : Certificat d'attribut.

II.2.2.2.8 Infrastructures à clés publiques (PKI)

Les PKI représentent l'ensemble d'algorithmes, protocoles et services qui protègent les échanges d'informations d'un réseau. Elles s'appuient notamment sur la cryptographie, la manipulation de la signature électronique et les certificats électroniques. Les PKI assurent des services de base tels que : (i) l'authentification d'une entité à travers l'utilisation des certificats, (ii) la confidentialité des échanges à l'aide des algorithmes de chiffrement utilisés par les protocoles (comme TLS par exemple), et (iii) l'intégrité des données grâce à l'utilisation des signatures numériques.

Dans la Figure II.19, nous présentons un schéma global des PKI où nous imaginons que deux utilisateurs, voulant communiquer en sécurité, deviennent des clients PKI en demandant des certificats afin d'être authentifiés dans le réseau. La CA est le cœur des PKI. C'est l'entité à qui toutes les entités du réseau font confiance. Elle est entourée par des entités comme : (i) la RA *(Registration Authority)*, une autorité d'enregistrement qui s'occupe de la réception des requêtes des utilisateurs, elle vérifie leur identité, demande leurs certificats à la CA et les publie dans des annuaires, (ii) une base de données pour stocker les certificats, (iii) les CRL *(Certificate Revocation List)* qui constituent la liste des certificats qui ont été révoqués et qui ne sont donc plus valables, (iv) un poste administrateur qui prend en main toute la gestion des PKI, et (v) des clients PKI qui souhaitent avoir des certificats et des services pour pouvoir communiquer entre eux en toute sécurité.

Quand un certificat n'est plus valide (expiration, renouvellement, etc.), il est révoqué par la CA. Cette dernière l'ajoute à la liste des certificats révoqués CRL. Cette liste sera

publiée, mise à jour régulièrement, et consultée lorsqu'une entité souhaite vérifier l'authenticité d'un certificat. Par contre, le délai de cette mise à jour pourra être exploité par un adversaire ou par le propriétaire lui-même. Par exemple, si un certificat vient d'être révoqué, l'entité possédant ce certificat pourra avoir un temps suffisant pour s'authentifier auprès d'une autre entité. Cette dernière, qui n'a pas eu le temps de consulter la nouvelle liste des révocations de certificats, accepte le certificat révoqué.

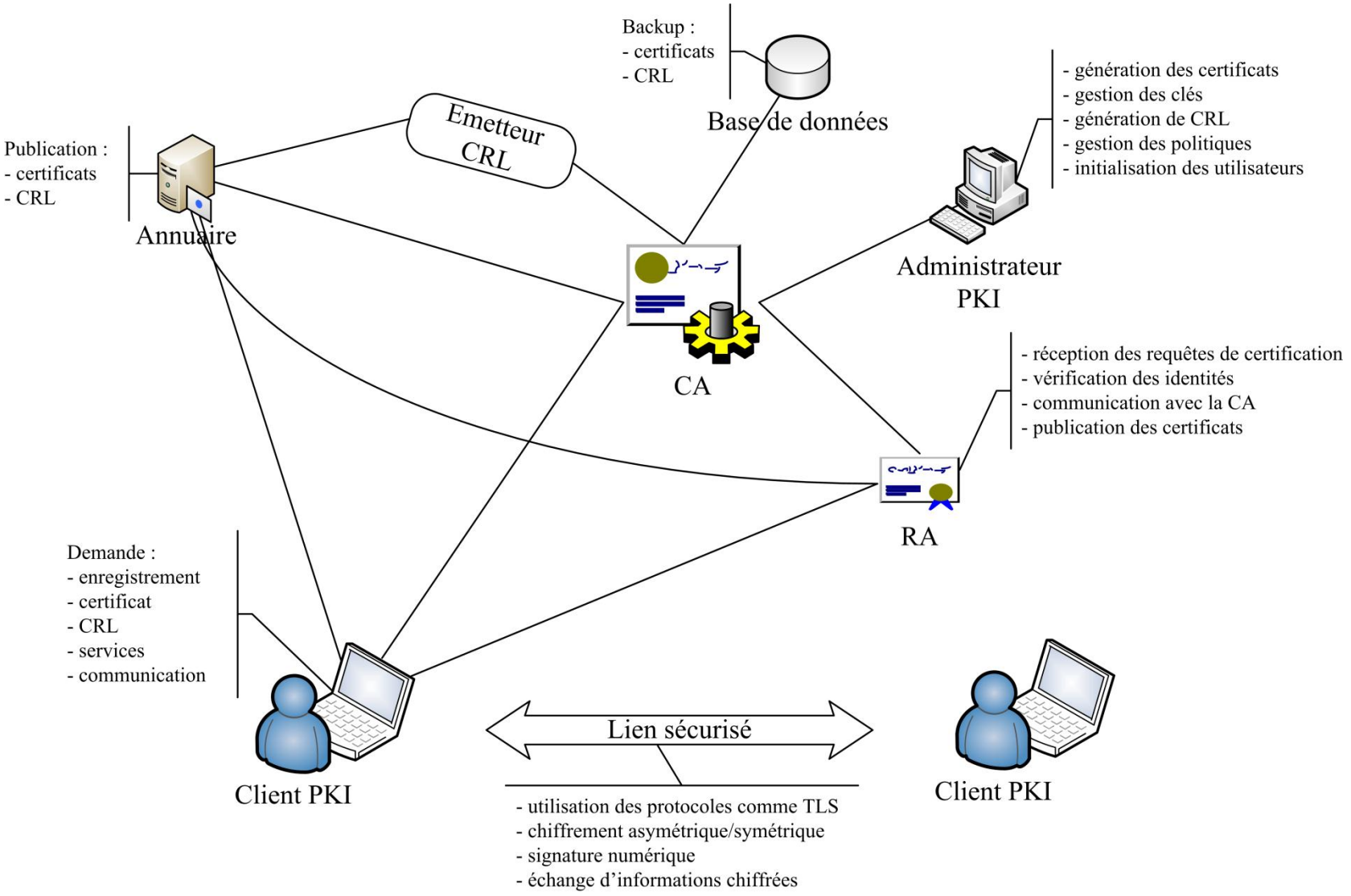


Figure II.19 : L'architecture des infrastructures à clés publiques (PKI).

II.2.2.2.9 Couplage ou « *Pairing* »

La cryptographie basée sur le couplage ou « Pairing » PBC *(Pairing-Based Cryptography)* est un domaine récent qui permet de réaliser des schémas cryptographiques radicalement nouveaux afin de rendre les protocoles cryptographiques plus efficaces. L'idée de couplage, comme la méthode de « *Weil Pairing* », a été introduite dans un premier temps dans le contexte de la cryptanalyse [63]. Cependant, la première application du couplage dans la cryptographie a été exploitée dans les années 2000 par les travaux de *Sakai* [64] et *Joux* [65].

Dans [65], les auteurs proposent une méthode pour que trois parties du réseau puissent partager une seule clé secrète en utilisant le principe de partage de DH (pour *Diffie* et *Hellman*) conçu initialement pour deux parties.

Au début, nous définissons dans l'Équation II.10 la bilinéarité, la propriété la plus importante sur laquelle se base le couplage. Soient l un entier positif, G_1et G_2 deux groupes additifs d'ordre l avec l'élément d'identité $\mathcal{O}$ et G_T un groupe multiplicatif d'ordre l avec l'élément d'identité 1.

Une fonction bilinéaire ê sur (G_1, G_T) est définie par $\hat{e} : G_1 \times G_2 \rightarrow G_T$.

$\forall P \in G_1, \forall Q \in G_2$ ***et*** $\forall a, b \in \mathbb{Z}^*$, ***nous avons***

$$\hat{e}(aP, bQ) = \hat{e}(P, bQ)^a = \hat{e}(aP, Q)^b = \hat{e}(P, Q)^{ab}.$$

Équation II.10 : La bilinéarité d'une fonction.

Le concept du couplage repose sur l'utilisation des fonctions bilinéaires afin : (i) d'accélérer le calcul de création de clés, (ii) de faire évoluer des protocoles pour les rendre plus généraux comme dans le cas de DHP, et (iii) de faciliter la distribution de clés dans les schémas de la cryptographie basée sur l'identité des nœuds [66].

La difficulté de la bilinéarité du DHP nommée BDHP repose sur le calcul de $\hat{e}(P, P)^{abc}$ étant donné P, aP, bP, cP. La puissance de BDHP repose sur la difficulté de résoudre le protocole DHP. Ainsi, trois parties du réseau peuvent partager une seule clé secrète en parallèle. Cette méthode peut être généralisée à *l-parties* en utilisant une fonction analogue *(*$\hat{e}_n$: $G_1^{l-1} \rightarrow G_T$*)* à celle de BDHP. Mais cette généralisation ne présente pas un grand intérêt sachant que même si elle résiste aux attaques passives, elle nécessite des tours

supplémentaires pour résister aux attaques actives. De plus, les auteurs de [67] montrent qu'il est difficile de trouver des fonctions multilinéaires pour $l > 3$.

II.2.2.3 Cryptographie hybride

La cryptographie hybride est un système qui réunit les deux systèmes de cryptographie : asymétrique et symétrique. La cryptographie asymétrique est beaucoup plus lente que la cryptographie symétrique. Pour des applications où le besoin est d'échanger beaucoup de données, le chiffrement asymétrique est inutilisable en pratique, parce qu'il consomme beaucoup de temps et de ressources. De ce fait, ce dernier est utilisé seulement pour échanger une clé secrète afin d'utiliser un chiffrement symétrique à la suite.

Plusieurs protocoles fonctionnent de nos jours sur le principe de cryptographie hybride. Nous citons parmi eux PGP *(Pretty Good Privacy)* [68] et le TLS [14].

II.3 Solutions adaptées aux communications des RCSF

Dans la partie II.2, nous avons détaillé les attaques qui menacent les réseaux sans fil et les RCSF en particulier. Pour y faire face, les chercheurs ont essayé de transposer le savoir-faire des réseaux filaires aux RCSF (les algorithmes de chiffrement, les méthodes d'authentification et de gestion de clés, etc.). Ce savoir-faire a fait preuve de ses performances dans les réseaux filaires grâce à l'utilisation de méthodes standardisées. Les solutions proposées dans la littérature utilisent des méthodes et des standards d'une manière qui ne répond pas complètement aux besoins des RCSF, cela en raison des capacités limitées en ressources énergétiques, de mémoire et de calcul de leurs nœuds.

Les RCSF sont menacés facilement à cause de l'utilisation de l'air comme médium de transmission et ont besoin d'être protégés par des méthodes adaptées à leurs applications. Les moyens de protection comme les algorithmes de chiffrement intégrés dans les protocoles de sécurité sont suffisamment sécurisés pour être utilisés dans les RCSF mais l'effort d'adaptation représente un grand challenge. L'adaptation de n'importe quel mécanisme de sécurité aux RSCF doit respecter les conditions suivantes : (i) l'échange de clés et des informations qui concernent leur création doit être réalisé rapidement puisque les communications passent via les signaux radio et risquent d'être interceptées à tout moment, (ii) la nature du déploiement des RCSF pour certaines applications rend les nœuds vulnérables à une capture physique (la perte de ces nœuds ne doit pas affecter la sécurité des

communications de tout le réseau), (iii) les nœuds d'un RCSF sont connus par être limités en ressources énergétiques, en capacité de calcul et de stockage. Ils le sont aussi en débit pour économiser leurs énergies. L'architecture proposée pour une application donnée doit trouver un compromis entre le niveau de sécurité demandé et le coût total de consommation de ressources. Par exemple, l'utilisation d'une seule clé de chiffrement pour tout le réseau permet de diminuer l'occupation mémoire, de supprimer le calcul de clés dérivées et ainsi d'économiser de l'énergie. Cependant, le chiffrement à l'aide de cette clé permet d'assurer la confidentialité des communications pour une courte durée seulement. Un adversaire peut extraire la clé à partir d'un nœud capturé physiquement et déchiffrer toutes les informations qui circulent dans le réseau. D'où l'intérêt d'avoir une architecture suffisamment sécurisée tout en respectant la limitation des ressources des nœuds.

L'établissement d'une clé secrète entre deux nœuds ou plusieurs est l'un des services de sécurité le plus important qui assure la confidentialité et l'intégrité des échanges dans un RCSF. Afin d'atteindre ce but d'une façon sécurisée, nous avons besoin d'un protocole de gestion de clés qui permet de gérer : (i) la pré-distribution de clés avant le déploiement, (ii) l'établissement de clés d'une façon sécurisée après le déploiement, et (iii) la délivrance, le renouvellement et la révocation des clés pendant toute la durée de vie du réseau. Ce protocole doit fournir un service aussi proche que possible des PKI d'un réseau filaire. La dynamicité de la gestion des clés permettra au réseau de rester évolutif et de pouvoir gérer les arrivées tardives de certains nœuds ou le départ d'autres. Le protocole doit non seulement intégrer rapidement les nouveaux arrivés en leur délivrant des clés certifiées, mais aussi révoquer les clés des anciens nœuds menacés ou enlevés pour une maintenance.

La méthode la plus simple pour chiffrer les échanges entre les nœuds en se basant sur les systèmes symétriques est de partager la même clé entre les nœuds du réseau. Cette clé pourra être renouvelée périodiquement pour lutter contre les agressions et les écoutes. Cependant, si un adversaire réussit à récupérer cette clé en la calculant ou en capturant un nœud, il pourra déchiffrer tous les échanges et mettre la sécurité de tout le réseau en danger.

Certaines méthodes sont basées sur les mécanismes des systèmes asymétriques notamment le concept des PKI des réseaux filaires, par exemple, en utilisant la station de base comme entité tierce de confiance pour remplacer la CA des PKI. La station de base sera chargée de l'authentification et de la distribution de clés authentifiées après le déploiement.

La difficulté de ces méthodes est de mettre en place ces mécanismes en tenant compte des limitations de bande passante du réseau, et d'énergie, de mémoire et de stockage des nœuds.

Il existe dans la littérature des propositions basées sur le système de chiffrement symétrique, le système asymétrique et le système hybride (qui utilisent les deux autres à la fois). Nous avons fait une classification qui tient compte des différents types de solutions.

II.3.1 Classification de méthodes et protocoles

La plupart des méthodes basées sur les systèmes symétriques, asymétriques ou hybrides résolvent le problème d'établissement de clés en passant par une phase de pré-distribution. La pré-distribution des clés de chiffrement dans un RCSF est le fait de stocker ces clés dans la mémoire des nœuds avant le déploiement. Nous trouvons plusieurs classifications de gestion et distribution de clés dans la littérature comme celle de [69]–[71]. Certaines classifications des méthodes sont basées sur le type d'approche comme le partage d'une clé entre deux nœuds *(Pair-wise)* ou plusieurs nœuds *(Group-wise)*, et d'autres, sur le fait qu'elles exploitent les probabilités, l'analyse combinatoire, etc. Nous avons choisi de faire une classification qui englobe l'ensemble des schémas de gestion et de distribution de clés en les regroupant en deux grandes familles. La première famille contient les schémas asymétriques et la deuxième regroupe les schémas symétriques. La Figure II.20 illustre cette classification. Dans la suite, nous détaillerons les principaux schémas de cette figure.

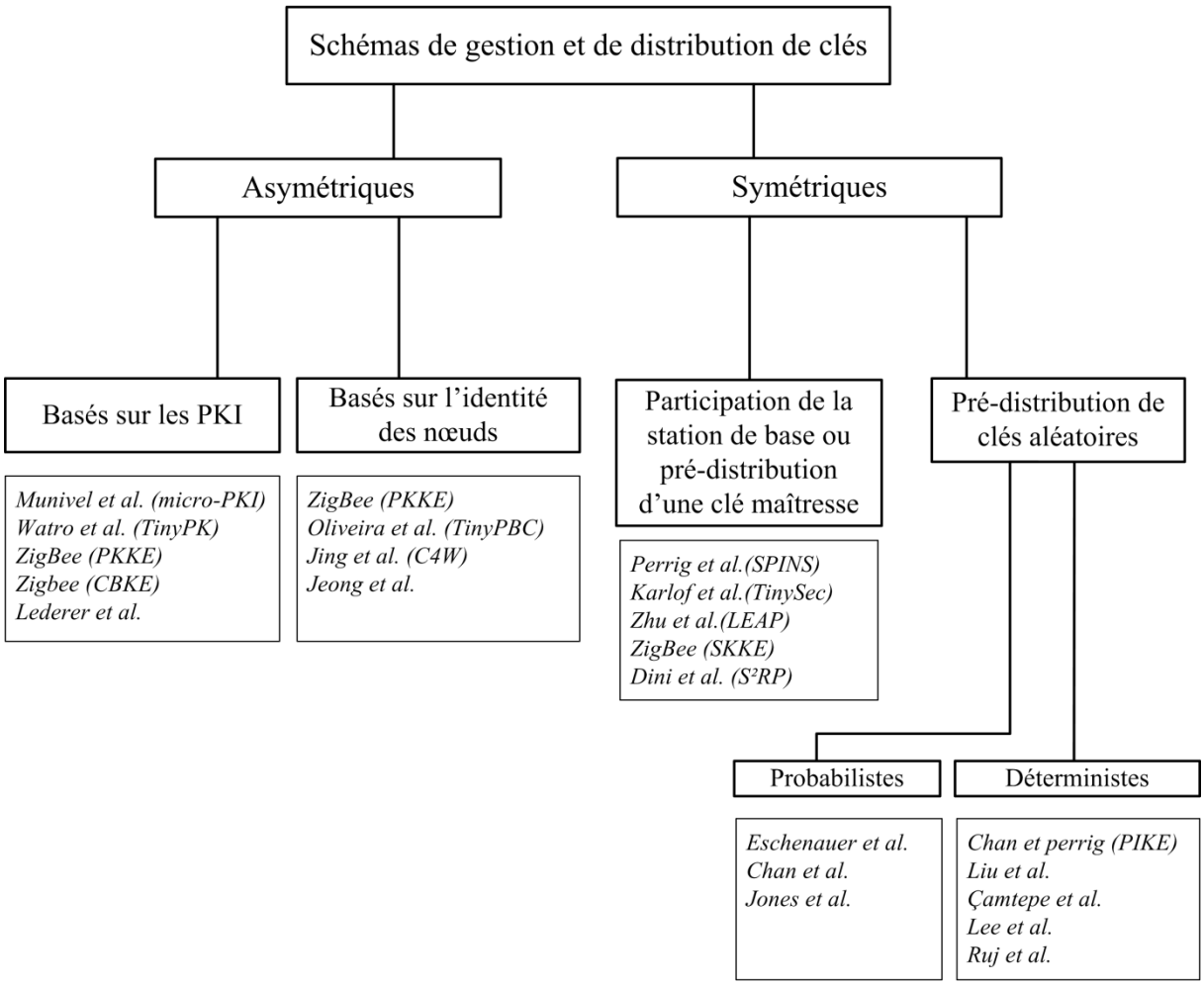


Figure II.20 : Classification des schémas de gestion et de distribution de clés.

II.3.2 Schémas symétriques

Les schémas de cette catégorie utilisent les mécanismes des systèmes symétriques dans le but d'établir une clé commune entre deux nœuds d'un RCSF. Cela est réalisé en trois étapes :

- *Pré-distribution de clés* (*Key predistribution*) : les clés stockées dans la mémoire avant le déploiement constituent le porte-clés *(key ring)* du nœud. S'il existe une clé commune entre deux nœuds, ils peuvent alors créer un lien sécurisé entre eux.
- Découverte de la clé commune *(Shared-key discovery)* : le protocole de communication après le déploiement est chargé de découvrir la clé commune entre deux nœuds voisins.
- Etablissement ou constitution d'un chemin sécurisé par des clés *(path-key establishment)* : s'il n'existe pas de clé commune entre deux nœuds voulant

communiquer, il faut alors trouver un chemin sécurisé entre eux. Ce chemin passe par un ensemble de nœuds qui contient déjà des liens sécurisés. Une fois ce chemin établi, les deux nœuds peuvent l'utiliser pour communiquer en toute sécurité.

Nous présentons dans la suite les schémas symétriques selon la décomposition de la Figure II.20.

II.3.2.1 Schémas de pré-distribution de clés aléatoires

Parmi les méthodes de pré-distribution de clés aléatoires, nous citons : (i) les méthodes probabilistes, définies par un tirage aléatoire (effectué par l'administrateur du réseau avant le déploiement) de trousseaux de clés choisies à la base à partir d'une grande liste sachant qu'il existe des clés communes entre ses trousseaux, avec une certaine probabilité, (ii) les méthodes déterministes, dans ce cas la liste de clés et les trousseaux de clés sont générés d'une façon déterministe pour s'assurer de l'établissement de certains liens entre les nœuds, et (iii) les méthodes hybrides, nées à partir d'une combinaison des deux méthodes précédentes pour augmenter la connectivité entre les nœuds du réseau et les adapter au passage à l'échelle.

II.3.2.1.1 Schémas probabilistes

Eschenauer et al. [72] ont été parmi les premiers à introduire l'idée de pré-distribuer ou de stocker dans les nœuds, avant le déploiement, un ensemble de clés choisies aléatoirement à partir d'un grand ensemble de clés. Cela à condition que deux nœuds voisins puissent avoir au moins une clé commune entre eux avec une certaine probabilité après le déploiement. Cette méthode est considérée comme le schéma basique des méthodes probabilistes. La phase de pré-distribution de clés du schéma est réalisée en plusieurs étapes. Elle commence par la génération d'un large ensemble de clés *P (Pool)* avec leurs identifiants. Ensuite, un ensemble de clés *k* est choisi aléatoirement dans *P* servant de porte-clés *(key ring)* à chaque nœud. Les porte-clés sont stockés ensuite dans la mémoire des nœuds. L'association entre la liste d'identifiant des porte-clés et l'identifiant des nœuds est stockée dans un nœud de contrôle. Un petit nombre de clés stockées dans les nœuds pourra assurer l'établissement d'une clé commune entre deux nœuds après le déploiement. Par exemple, pour une probabilité d'appairage de 0,5, 75 clés choisies aléatoirement à partir de 10 000 clés sont suffisantes pour former n'importe quel porte-clés. La deuxième phase de cette méthode est la découverte de clés communes. Afin de découvrir si un nœud voisin partage une clé commune, un nœud source diffuse la liste des identifiants de son porte-clés en clair à ses voisins. Même

si l'adversaire découvre cette liste, il n'aura pas l'opportunité de découvrir les clés de la liste et par la suite il n'aura pas la clé partagée utilisée pour le chiffrement. Tandis que s'il capture le nœud, il pourra découvrir la clé partagée et déchiffrer les communications du lien établi.

Lorsqu'un nœud est capturé, son porte-clés doit être révoqué. Un nœud de contrôle (qui pourra être mobile) diffuse un message de révocation signé contenant la liste des identifiants du porte-clés révoqué. Afin de signer cette liste, le nœud de contrôle génère une clé de signature K_e, la chiffre avec K^{ci} (clé partagée entre tous les nœuds et $i^{ème}$ nœud de contrôle avant le déploiement) et l'envoie individuellement *(Unicast)* aux nœuds. Les nœuds sont capables de vérifier alors la signature du message reçu. Chaque nœud localise les identifiants communs avec cette liste et supprime les clés correspondantes de leur mémoire. Une fois supprimés, des liens créés peuvent disparaitre et affectent l'envoi des données. Les nœuds affectés peuvent reconfigurer les liens disparus en commençant de nouveau la phase d'établissement de clés avec leurs voisins. La révocation affecte seulement quelques nœuds et une petite partie des clés de leur porte-clés sera supprimée. Cela est nécessaire pour limiter la connectivité du nœud compromis.

Dans ce schéma, le renouvellement de clés d'un nœud est équivalent à une auto-révocation de la clé partagée avec un autre nœud. Le nœud recommence la phase de découverte de la clé commune avec son voisin sans impliquer aucun nœud de contrôle dans l'opération. Notons qu'après chaque opération de renouvellement ou de révocation, un nœud perd au moins une clé de son porte-clés et risque d'avoir peu de liens avec ses voisins, voire aucun en cas d'épuisement de tout le porte-clés.

Chan et al. [73] se sont basés sur la méthode de *Eschenauer et al.* afin de proposer un nouveau schéma de pré-distribution où deux nœuds voisins doivent partager plusieurs clés au lieu d'une seule. Cela augmente la résistance contre la capture de nœuds car l'adversaire aura plus de difficulté à trouver la clé établie avec un nœud voisin parmi les clés communes.

Jones et al. [74] proposent une architecture de sécurité en utilisant une méthode de distribution probabiliste de clés basée sur un découpage du réseau en secteurs. Cette solution est applicable sur les réseaux dont les nœuds sont anonymes et n'ont pas de position prédéterminée. La configuration est dynamique et capable de supporter un changement d'applications et de topologie. Aucun nœud ne connaît la topologie du réseau. Les auteurs supposent que la station de base se trouve au centre du réseau. Le but est de faire une distribution de clés symétriques qui conduit à l'établissement de chemins sécurisés de bout-

en-bout entre un émetteur et un récepteur. La pré-distribution de clés se base sur l'utilisation du schéma basique de *Eschenauer et al.*. Avant le déploiement, chaque capteur est chargé avec un ensemble de clés *m*, sélectionnées aléatoirement à partir d'un ensemble de clés *k*. Le nombre de clés |*k*| est choisi de manière à ce que deux trousseaux aléatoires de taille |*m*| puissent avoir au moins une clé commune avec une probabilité *p*. Le nombre de clés communes entre deux trousseaux pourra être un paramètre de sécurité à préciser dans les implémentations.

Après le déploiement, la station de base découpe le réseau en secteurs indépendants et en cercles définissant plusieurs sauts *(mutli-hop)*. La Figure II.21 présente le réseau découpé en cercles et secteurs tout en montrant les clés symétriques correspondant au secteur . La station de base distribue secrètement (voir Annexe VI.2) aux nœuds de chaque secteur W_i ($0 < i < n$, où *n* est le nombre total de secteurs) la clé du secteur WK_i. Le nœud source (ou émetteur) diffuse aux nœuds voisins un index contenant sa liste de clés m. Il commence à établir des liens de confiance avec les nœuds voisins, qui à leur tour, diffusent leur liste à leurs voisins afin de trouver des chemins de confiance jusqu'à la station de base. Une fois le chemin de confiance établi entre le nœud source et la station de base, le nœud source génère une clé de chemin PK_j (*Path-Key*) et l'envoie sur chacun des liens j établis au sein du même secteur ($0 < j < l$, où l est le nombre total des chemins établis au sein d'un secteur). La clé PK_j est chiffrée à l'aide de la clé WK_i du secteur pour que seulement les membres du secteur puissent la déchiffrer.

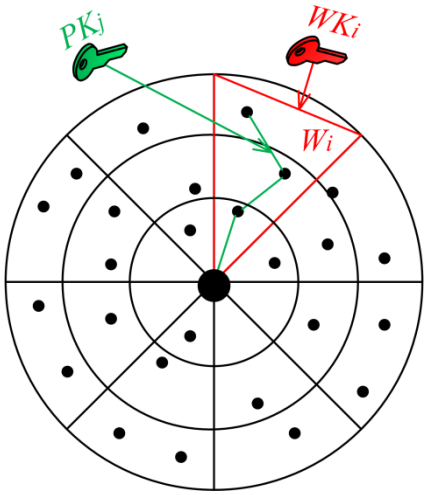


Figure II.21 : Découpage du réseau en secteurs. Chaque secteur i possède une clé symétrique WK_i et chaque chemin de confiance établi j possède une clé symétrique PK_j.

Les méthodes probabilistes sont simples dans leur mise en œuvre et ne sont pas coûteuses dans la phase d'établissement de clés après le déploiement. Par contre, elles souffrent de quelques désavantages. D'une part, la taille de l'ensemble de clés pré-distribuées va augmenter avec la taille du réseau et la mémoire des nœuds sera chargée. D'autre part, la méthode ne peut pas protéger le réseau en cas de capture physique de nœuds. Une fois que l'ensemble de clés d'un nœud est extrait, l'adversaire pourra trouver les clés communes établies avec ces voisins. De plus, lorsque le graphe du réseau est bien connecté, un petit nombre de nœuds compromis affecte un grand nombre de nœuds du reste du réseau.

II.3.2.1.2 Schémas déterministes

Dans cette partie, nous présentons des schémas de pré-distribution déterministes. Ces distributions ont l'avantage de garantir des graphes complètement connectés car elles permettent à chaque nœud d'établir une clé unique avec n'importe quel autre nœud du réseau. En revanche, ceci nécessite une grande capacité de stockage. Chaque clé distribuée est unique et est établie entre seulement deux nœuds, elle est connue sous le nom de « *pairwise key* ». Il existe plusieurs schémas pour distribuer des clés symétriques d'une manière déterministe. Nous pouvons les classer en trois catégories : méthodes basées sur l'utilisation de grilles *(Grid-based schemes)*[75][76], méthodes basées sur des structures combinatoires [77][78], et méthodes basées sur l'utilisation des codes [79]. Nous avons limité la synthèse bibliographique de cette partie à quelques méthodes de la première catégorie.

Chan et Perrig [75] ont proposé une méthode qui s'appelle PIKE *(Peer Intermediaries for Key Establishment)* dédiée à l'établissement de clés. Elle fait partie des méthodes qui utilisent des grilles pour faire la pré-distribution de clés. La plupart des schémas probabilistes réalise leur distribution par un stockage des clés d'ordre $\mathcal{O}(N)$ (où N est le nombre maximum de nœuds dans le réseau) dans les mémoires des nœuds. De plus, ils utilisent la station de base comme nœud intermédiaire de confiance. Cela entraîne un épuisement d'énergie des nœuds proches de la station de base vu que cette méthode impose que beaucoup de messages passent à travers eux.

Afin de résoudre les problématiques liées au stockage d'un grand nombre de clés et à l'épuisement de l'énergie, PIKE propose une méthode d'établissement de clés déterministe qui pourra se réaliser en stockant des clés d'ordre $\mathcal{O}(\sqrt{N})$ dans la mémoire des nœuds. Cette méthode utilise des nœuds ordinaires du réseau comme intermédiaires de confiance. Les clés

sont pré-distribuées pour que n'importe quel couple de nœuds A et B puisse toujours avoir la possibilité de trouver un nœud C du réseau qui partage une clé unique avec A et B. Alors A pourra utiliser C comme nœud intermédiaire de confiance afin de transmettre son message d'établissement de clé à B.

Les identifiants (ID) des nœuds sont arrangés dans une structure en grille carrée de dimension $\sqrt{N} \times \sqrt{N}$. Chaque nœud aura un ID de la forme (x, y) où $x, y \in \{0, 1, 2, \dots, \sqrt{N} - 1\}$. Chaque nœud sera seulement chargé d'une clé secrète unique avec chacun des nœuds des deux listes : (i, y) où $\forall i \in \{0, 1, 2, \dots, \sqrt{N}\}$ et (x, j) où $\forall j \in \{0, 1, 2, \dots, \sqrt{N}\}$. Par exemple, (x, y) partagera une clé unique $K_{(x,y),(1,y)}$ avec $(1, y)$ et une clé différente avec $(2, y)$ et ainsi de suite. Afin de mieux comprendre le fonctionnement de la méthode, nous reprenons un exemple fourni par les auteurs de PIKE. La Figure II.22 présente une grille logique (ou espace virtuel) d'identifiants de 100 nœuds. Chaque nombre présent dans la grille est l'identifiant d'un nœud du réseau. Cela permet de constater que les nœuds 91 et 14 peuvent partager une clé unique avec chacun des nœuds qui appartiennent à leur colonne ou à leur ligne. Ainsi, les nœuds 11 et 94 peuvent jouer le rôle d'intermédiaires de confiance dans l'établissement de clés entre 91 et 14.

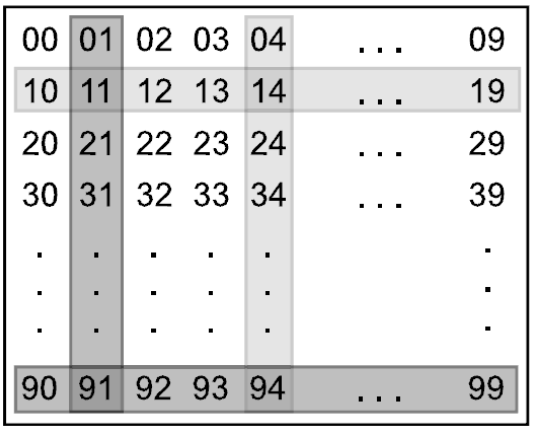


Figure II.22 : Espace virtuel d'identifiants de nœuds d'un réseau de 100 nœuds [75].

Chaque nœud de la Figure II.22 sera chargé avec 18 clés (9 clés pour les nœuds appartenant à sa ligne et 9 clés pour les nœuds appartenant à sa colonne). Pour généraliser, chaque nœud d'un réseau de taille *N* est chargé avec $2(\sqrt{N} - 1)$ clés distinctes. Une fois l'intermédiaire de confiance C choisi (plus proche géographiquement de A et B), A chiffre une nouvelle clé symétrique à partager avec B à l'aide de sa clé commune avec C et l'envoie à C. A la réception, C déchiffre le message et récupère la clé. Ensuite, C la chiffre de nouveau avec la

clé commune partagée avec B avant de l'envoyer à B. Finalement, B reçoit la clé partagée avec A en déchiffrant à l'aide de la clé unique partagée avec C et envoie un acquittement de réception à A. La Figure II.23 illustre cet établissement de clés. Chaque message est accompagné d'un message d'authentification MAC (code) afin de garantir l'intégrité des messages échangés.

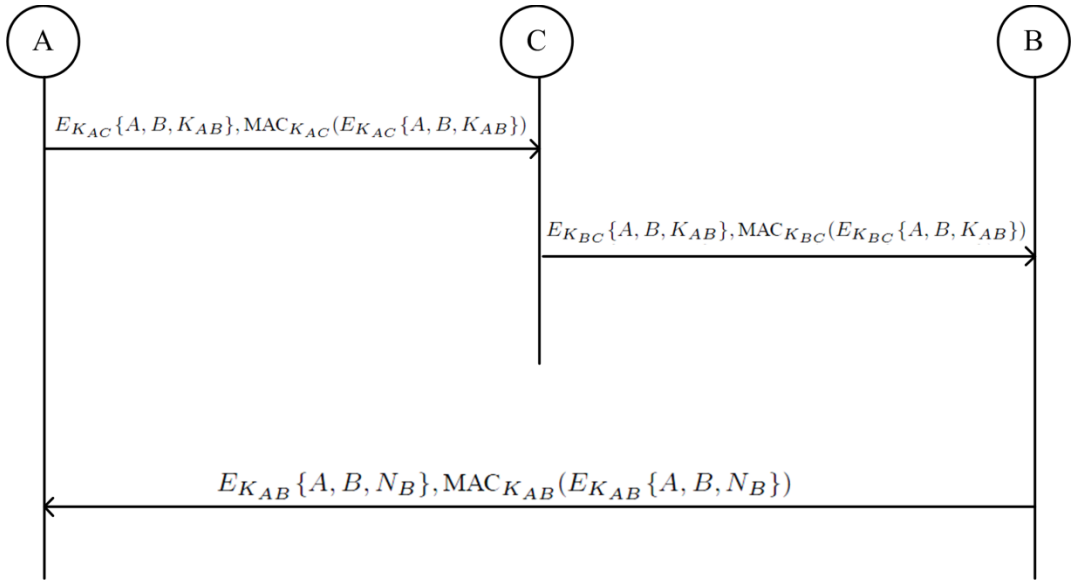


Figure II.23 : Établissement de clés entre A et B en passant par l'intermédiaire de confiance C.

Notons que PIKE minimise le stockage de clés dans la mémoire des nœuds avant le déploiement par rapport à d'autres méthodes. Cependant, les échanges de messages d'établissement de clés consomment du temps et de l'énergie. PIKE a proposé l'utilisation des nœuds du réseau comme intermédiaires de confiance à la place de la station de base. Cela est proposé afin de ne pas épuiser les nœuds proches de la station de base. Cependant, cette solution pourrait avoir un désavantage, car si les nœuds intermédiaires de confiance de A et B sont capturés, A ne pourra plus partager une clé avec B.

Liu et al. [76] ont proposé une approche similaire à *Chan et Perrig* [75] du fait qu'elle utilise aussi des grilles logiques. Par contre, leurs travaux s'appuient sur la localisation des nœuds déployés en les divisant en groupes. Chaque ensemble de nœuds est déployé afin de former un groupe de déploiement *(deployment group).* Les nœuds du réseau déployé sont divisés en *n* groupes contenant chacun un nombre *m* de nœuds. Le schéma de pré-distribution pour établir une clé partagée unique *(pairwise key)* entre deux nœuds d'un même groupe est appelé *« in-group predistribution »*. Tandis que le schéma de pré-distribution pour établir une clé partagée unique entre deux groupes est appelé *« cross-group predistribution »*. Le premier

schéma *(in-group)* pourra alors être n'importe lequel des schémas existants de pré-distribution. $\{D_i\}_{i=1,2,\dots,n}$ sont définis comme étant des instances d'un schéma de pré-distribution existant dédié à des *in-group* $\{G_i\}_{i=1,2,\dots,n}$. Alors que $\{D'_i\}_{i=1,2,\dots,m}$ sont définis comme étant des instances d'un schéma de pré-distribution dédiées à des *cross-group* $\{G'_i\}_{i=1,2,\dots,m}$. Ce schéma doit respecter les deux conditions suivantes : (i) chaque G'_i contient exactement un seul nœud d'un groupe de déploiement et (ii) $\forall\ i,\ j$ avec $i \neq j$, on a $G'_i \cap G'_j = \emptyset$ et $|G'_i \cap G_j| = 1$. Ainsi, chaque *cross-group* constitue un lien potentiel pour n'importe quel couple de deux *in-group*.

La Figure II.24 montre un exemple de quatre *in-group* constituant les colonnes de la figure et de trois *cross-group* constituant ses lignes. Les numéros de la figure sont les identifiants des nœuds déployés. Une fois déployés, deux nœuds appartenant à un même *in-group* G_i (une même colonne) peuvent établir un lien direct entre eux en suivant les règles d'établissement de l'instance D_i. Deux nœuds appartenant à un même *cross-group* G'_i (une même ligne) peuvent aussi établir un lien entre eux en suivant les règles d'établissement de l'instance D'_i. Après cette étape, les groupes peuvent établir des liens entre eux en créant des ponts. Un pont entre deux groupes G_i et G_j est défini par une paire de nœuds $\langle a, b\rangle$ ($a \in G_i$, $b \in G_j$) appartenant à un même *cross-group* ($a, b \in G'_k$). Dans l'exemple de la Figure II.24, nous trouvons trois ponts entre G_1et G_4 : $\langle 1{,}10\rangle, \langle 2{,}11\rangle$ et $\langle 3{,}12\rangle$. Les ponts vont servir dans la création des clés de chemins (de chemins sécurisés par des clés). Par exemple, si le nœud 1 souhaite partager une clé unique avec le nœud 12, il essaie de passer par l'intermédiaire du pont $\langle 1{,}10\rangle$. S'il ne réussit pas, il essaie ensuite de passer par le pont $\langle 3{,}12\rangle$, et s'il ne réussit pas non plus, il essaie de passer par le pont $\langle 2{,}11\rangle$. Au cas où aucun pont ne donne satisfaction, l'étape d'établissement de clés de chemin entre 1 et 12 échoue.

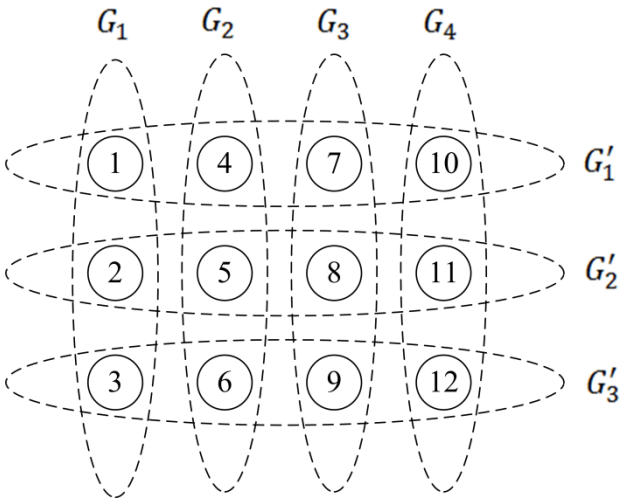


Figure II.24 : Exemple de construction de groupes.

II.3.2.2 Schémas basés sur la participation de la station de base ou la pré-distribution d'une clé maîtresse

Perrig et al. [80] ont proposé le protocole SPINS, l'un des premiers protocoles de sécurité proposés pour les RCSF. Il est basé sur deux protocoles : SNEP *(Sensor Network Encryption Protocol)* et µTESLA. SNEP assure la confidentialité et l'authentification des données entre deux nœuds avec un faible coût (ajout de 8 octets à chaque message). Tandis que µTESLA, version étendue de TESLA [81], assure l'authentification de la diffusion. SNEP repose sur l'utilisation de l'algorithme de chiffrement RC5 [82] en l'employant en mode de compteur CTR (*CounTeR*). La topologie du réseau autorise au début du déploiement uniquement les communications entre les nœuds capteurs et la (les) station(s) de base. SPINS présente une méthode pour étendre la confiance établie entre les nœuds et la station de base aux liens établis entre les nœuds directement.

Le protocole consiste à stocker une clé secrète maîtresse (X) dans le nœud capteur et dans sa station de base. Après le déploiement, un nœud A et une station de base B dérivent chacun deux clés secrètes à partir de la clé maîtresse ($X_{AB} = X_{BA}$). Le nœud A obtient K_{AB} et K'_{AB}, et le nœud B obtient K_{BA} et K'_{BA}. La première clé est utilisée pour le chiffrement et la deuxième est dédiée au calcul du code d'authentification des messages (MAC). La dérivation se fait à l'aide d'une fonction pseudo-aléatoire F. Puisque les deux nœuds partagent la même clé maîtresse et la même fonction de dérivation, chaque nœud pourra dériver les clés secrètes de l'autre pour pouvoir déchiffrer et vérifier les messages reçus. Au

lieu d'envoyer les compteurs avec les messages, A et B partagent deux compteurs (un pour chaque direction de la communication). Ils sont synchronisés à l'aide d'un protocole d'échange de compteurs, illustré dans la Figure II.25. C_A et C_B sont les compteurs respectifs de A et B. Les deux compteurs ne sont pas secrets, alors A peut envoyer son compteur à B en clair. À la réception, B envoie son compteur accompagné du code d'authentification (MAC) de la combinaison des deux compteurs. Ainsi, A pourra vérifier l'intégrité de C_B. Il envoie ensuite le code d'authentification de la combinaison des deux compteurs à B afin qu'il puisse vérifier l'intégrité de C_A.

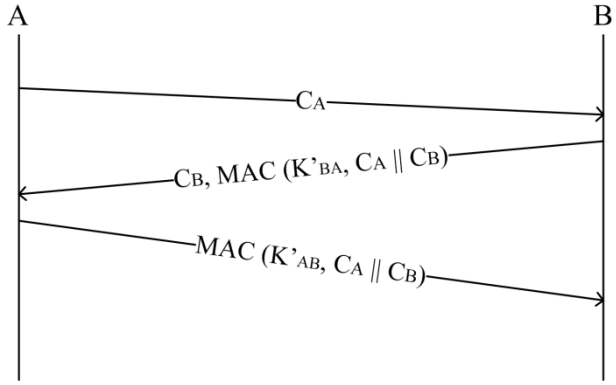


Figure II.25 : Protocole d'échange de compteurs.

La méthode de partage de compteur économise de l'énergie en remplaçant l'envoi des compteurs dans chaque message par un calcul de MAC (code) et une synchronisation de compteur. A noter que cette méthode ne pourra être bénéfique que si les deux nœuds communicants ont beaucoup de données à échanger. De plus, chaque communication a besoin de deux compteurs synchronisés (un par direction), et ceci pourrait surcharger le calcul et le stockage mémoire des nœuds.

Karlof et al. [33] proposent le protocole TinySec, la première implémentation complète d'une architecture sécurisée au niveau de la couche liaison de données pour les RCSF. Cette implémentation supporte deux options de sécurité : une authentification des messages avec chiffrement des données (TinySec-AE) et une authentification des messages sans chiffrement des données (TinySec-Auth). Comme SPINS, TinySec utilise des algorithmes standards de cryptographie afin de garantir la confidentialité et vérifier l'intégrité des messages. Les auteurs de TinySec trouvent que Skipjack [83] est un algorithme mieux adapté aux RCSF que RC5 (algorithme utilisé par SPINS). En effet, les évaluations de TinySec ont prouvé que RC5 a besoin d'un pré-calcul de clés utilisant 104 octets des RAM.

TinySec utilise le mode de chiffrement CBC *(Cipher Block. Chaining)* au lieu du mode CTR (utilisé par SPINS). En effet, le mode CTR pourra donner pour plusieurs chiffrements de paquets les mêmes nombres aléatoires. Ces nombres étant utilisés principalement à la fabrication des séquences de clés de chiffrement, leur répétition pourra affaiblir le niveau de sécurité de cette solution et par la suite permettre aux adversaires de découvrir le contenu des messages. TinySec étant plutôt une implémentation qu'une proposition de distribution de clés, il vient pour compléter une méthode de distribution de clés adaptée au réseau déployé. Deux nœuds ont besoin de deux clés symétriques partagées pour communiquer. La première est utilisée pour chiffrer les messages et la deuxième pour calculer les MAC (code) des messages.

Zhu et al. [84] proposent le protocole LEAP *(Localized Encryption and Authentication Protocol)*, un protocole d'authentification et de gestion de clés destiné aux RCSF. LEAP repose sur l'utilisation de quatre types de clés par nœud : (i) une clé individuelle partagée avec la station de base, (ii) une clé unique partagée avec un autre nœud, (iii) une clé de *cluster* partagée avec ses nœuds voisins du même *cluster* et, (iv) une clé de groupe, partagée avec tous les nœuds du réseau. Le contrôleur du réseau génère une clé maîtresse initiale K_I et la stocke dans la mémoire des nœuds avant le déploiement. Après le déploiement, chaque nœud *u* du réseau dérive de cette clé sa propre clé maîtresse $K_u = f_{K_I}(u)$. Puis il diffuse un message HELLO contenant son identifiant à ses voisins $(u \rightarrow * : u)$. Lorsqu'il reçoit un ACK d'un voisin *v* $(v \rightarrow u : v, MAC(K_v,\ u|v))$, il pourra vérifier son identité en calculant au début $K_v = f_{K_I}(v)$ et par suite le MAC. Une fois l'identité de *v* vérifiée, *u* calcule sa clé unique partagée avec lui $K_{uv} = f_{K_v}(u)$. Le nœud *v* pourra calculer cette clé de la même façon. Notons que *u* n'a pas besoin de s'authentifier auprès de *v* car l'authentification sera implicite lors de l'envoi des messages chiffrés à l'aide de la clé commune. Après un certain temps (fixé au début du déploiement), *u* supprime la clé maîtresse initiale (K_I) et les clés calculées de ses nœuds voisins (les K_v). Il garde seulement sa clé individuelle et les clés uniques partagées avec ses voisins. Supposons que *u* soit maintenant un chef de cluster qui souhaite envoyer une clé de cluster à ses nœuds voisins $v_1, v_2, \ldots, v_m$. Il génère au début une clé aléatoire K_u^c. Il la chiffre à l'aide de la clé partagée avec chaque v_i $(1 \leq i \leq m)$ avant de l'envoyer en unicast à v_i $(u \rightarrow v_i : (K_u^c)_{K_{uv_i}})$. Afin de distribuer des clés de groupe partagées par tout le réseau, LEAP adapte le protocole µTESLA (proposé par SPINS) et cela en utilisant le principe de chaîne de clés (expliqué dans la Figure II.27) et l'arbre recouvrant le réseau *(spanning tree)*. La station de base envoie K_g', une nouvelle clé de groupe, à tous ses fils de l'arbre en la

chiffrant à l'aide des clés de *cluster* déjà partagées. L'arbre recouvrant le réseau aide non seulement à délivrer les clés de groupe mais aussi à les renouveler en cas de révocation. Finalement, LEAP utilise l'algorithme RC5 pour le chiffrement et utilise le mode CBC-MAC pour l'authentification des messages.

Le standard *ZigBee* [5] est une pile protocolaire qui s'appuie sur les couches basses de la norme IEEE 802.15.4 dédiée aux réseaux WPAN *(Wireless Personal Area Networks)*. Dans la topologie d'un réseau *ZigBee*, nous pouvons différencier trois types de nœuds : le coordinateur, les routeurs et les capteurs. Un coordinateur peut jouer le rôle d'un centre de confiance TC *(Trust Centre)* qui est alors considéré comme application, plutôt qu'un nœud du réseau en raison de sa position particulière dans la topologie du réseau. Le TC a pour rôle de distribuer des clés et d'aider les nœuds dans l'étape d'établissement de clés. Il existe trois types de clés utilisées par un nœud : (i) la clé LK *(Link Key)*, une clé unique partagée avec un autre nœud du réseau, (ii) la clé NK *(Network Key)* partagée avec tout le réseau, et (iii) la clé MK *(Master Key)* utilisée dans l'établissement de la clé LK. Afin d'établir la clé LK entre deux nœuds du réseau, *ZigBee* propose le protocole SKKE [85] *(Symmetric-Key Key Establishment)* basé sur des mécanismes symétriques utilisant MK comme une information de confiance.

Nous présentons dans la Figure II.26 l'établissement de la clé LK en suivant le protocole SKKE entre deux nœuds U et V du réseau. Au début, U envoie un challenge QEU (séquence générée aléatoirement) à V. Ensuite, V génère à son tour son challenge QEV et l'envoie à U. À la réception de QEV, U calcule à l'aide de la clé MK les deux clés MacKey et KeyData. Le but est d'utiliser la clé MacKey afin d'authentifier la clé KeyData sans avoir besoin de l'envoyer. Pour cela, U construit le MacData2 (concaténation des identifiants et des challenges) et calcule son MAC à l'aide de la clé MacData, avant d'envoyer le résultat (MacTag2) à V. Lorsque V reçoit le MacTag2, il va pouvoir le vérifier en calculant le MacData2 et en le comparant au résultat. Une fois la vérification faite, V procède au calcul de MacKey et de KeyData (les mêmes clés calculées par U). En suivant la même procédure que U, V calcule le MAC de son MacTag1 et l'envoie à U. La réussite de la vérification chez U implique que U et V ont créé la même clé KeyData. Cette clé sera utilisée comme l'unique clé partagée LK entre U et V.

ZigBee a aussi proposé un protocole qui s'appelle MEA *(Mutual Entity Authentication)* pour garantir une authentification mutuelle entre deux nœuds. La procédure

d'authentification mutuelle ressemble à la procédure utilisée par SKKE sauf que les deux nœuds ne créent pas les deux clés MacKey et KeyData. Ainsi, le MacTag sera calculé à l'aide de NK au lieu de MacKey. Pour plus de détails sur les deux protocoles SKKE et MEA, veuillez-vous référer à [85], la spécification détaillée du standard *ZigBee*.

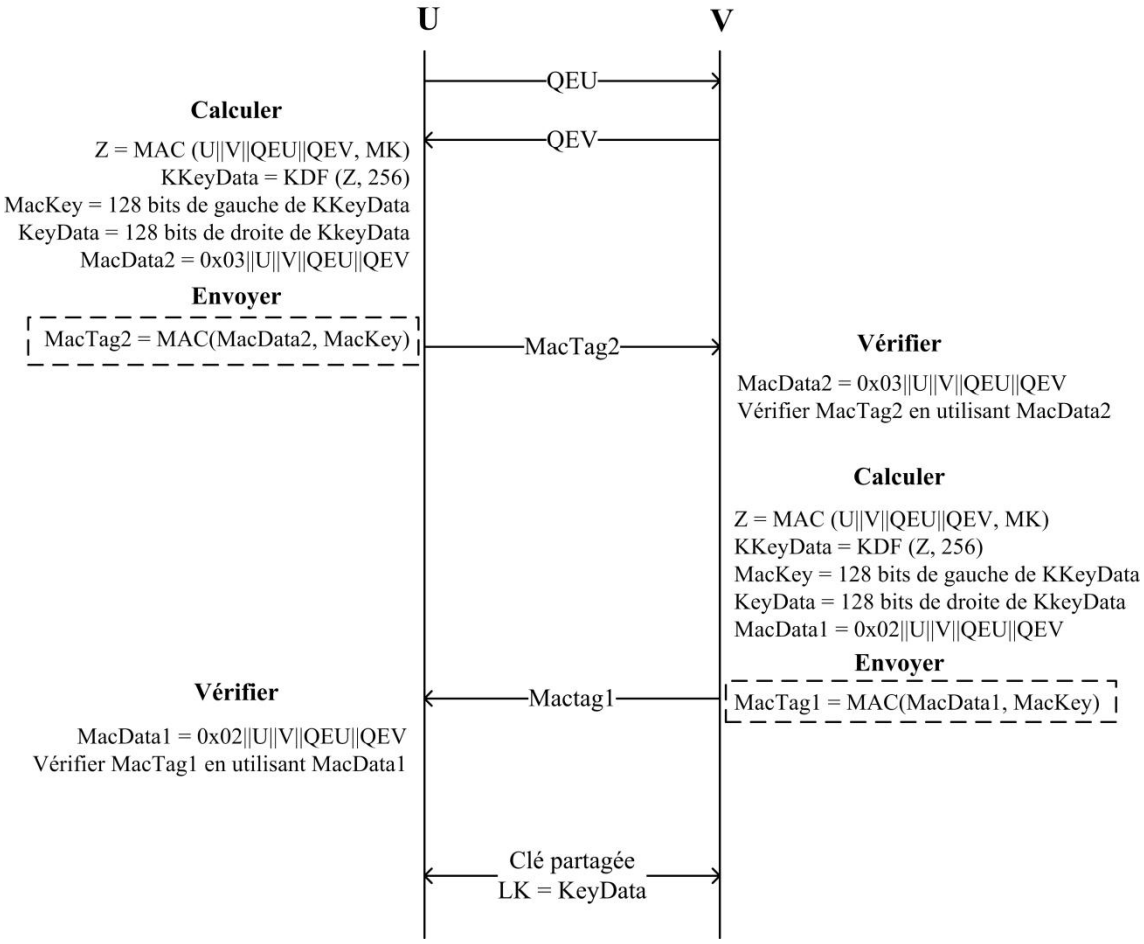


Figure II.26 : Le protocole symétrique SKKE de *ZigBee*. Partage d'une clé symétrique unique.

Plusieurs méthodes prévoient une gestion en cas d'arrivée tardive, de départ ou de capture de nœuds. La révocation et le renouvellement des clés peuvent avoir le même niveau d'importance que la distribution. *Dini et al.* [86] présentent un protocole de renouvellement et de révocation qui s'appelle S^2RP *(Secure and Scalable Rekeying Protocol).* Il garantit la distribution authentifiée de clés en utilisant un chiffrement symétrique et une fonction de hachage en se basant sur le modèle de *group communication* [87]. Pour assurer la sécurité des communications du groupe, S^2RP propose le renouvellement en respectant les propriétés suivantes : (i) *backward security*, lorsqu'un nœud souhaite rejoindre le groupe, il ne faut pas qu'il soit capable de décrypter les messages chiffrés avec une ancienne clé, et (ii) *forward*

security, lorsqu'un nœud quitte ou est obligé de quitter le groupe, il ne faut plus qu'il soit capable de communiquer avec les nœuds du groupe. Lorsqu'un nœud rejoint ou quitte le groupe, la clé symétrique courante de chiffrement doit être révoquée et renouvelée d'une façon qui permette à tous les membres du groupe de l'identifier. Pour ce faire, S^2RP intègre deux techniques. La première est LKH *(logical key hierarchy),* qui est une technique déjà utilisée dans les réseaux filaires. Elle permet de renouveler les clés d'un groupe avec un nombre de messages envoyés aux nœuds d'un ordre logarithmique par rapport au nombre des nœuds du réseau. La deuxième technique est une liste chaînée de clés ou *key-chain,* qui est une technique d'authentification utilisée comme méthode efficace pour l'authentification des clés dans les RCSF. Cette méthode s'appuie sur l'utilisation d'une fonction de hachage *(*OWHFs : *one-way hash functions)*. A l'initialisation, un émetteur S crée une chaîne de clés en utilisant une fonction de hachage H. Il envoie la tête de la chaîne $SK^{(0)}$ à un récepteur R à l'aide d'un canal sécurisé. Lorsque S envoie n'importe quelle clé de la chaîne, R est capable de faire un certain nombre de hachage pour tomber sur une clé déjà authentifiée. La Figure II.27 illustre le fonctionnement du mécanisme d'authentification de *key-chain* en prenant un exemple d'authentification de la clé $SK^{(j)}$.

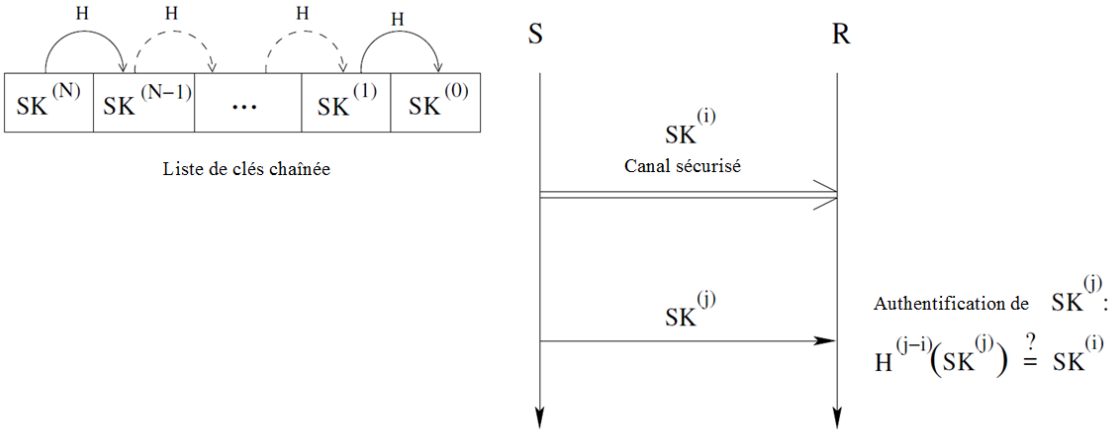


Figure II.27 : Scénario d'authentification d'une clé à partir d'une clé déjà partagée.

La liste de clés chaînée est exploitée dans le temps en parcourant le chemin de $SK^{(N)}$ vers $SK^{(0)}$ ce qui rend ce chemin inverse très difficile à retrouver. Notons qu'un nœud partant du groupe ne pourra pas calculer à partir d'une clé courante $K^{(i)}$ les prochaines clés (propriété de OWHF). Dans ce cas, la propriété *forward security* est assurée. Tandis que si un nouveau nœud rejoint le groupe et reçoit la clé courante, il est capable de calculer toutes les clés précédentes en appliquant la fonction de hachage un certain nombre de fois sur la clé

courante. Ainsi, il pourra décrypter tous les anciens messages communiqués et la propriété de *backward security* n'est plus assurée. Afin de résoudre ce problème, S^2RP propose une solution basée sur la distribution d'une clé nommée K_J ou *join-key*. Elle est partagée par tous les membres du groupe et elle est renouvelée secrètement lorsqu'un nouveau nœud rejoint le groupe. Tous les membres calculent la nouvelle clé courante du groupe K_{grp} en mixant K_J avec l'ancienne clé courante, $K_{grp} = M(K_J, K_i)$, où M est une fonction de mixage [86]. Le nouveau nœud reçoit K_{grp} et il ne pourra pas calculer les anciennes clés parce qu'il ne possède pas les anciennes clés K_J. Par conséquent, la propriété de *backward security* est assurée.

II.3.3 Schémas asymétriques

Les schémas de cette catégorie utilisent les mécanismes des systèmes asymétriques dans le but d'établir une clé commune entre deux nœuds ou un groupe de nœuds d'un RCSF. Nous présentons dans la suite ces schémas selon la décomposition présentée dans la Figure II.20.

II.3.3.1 Basées sur les PKI

Munivel et al. [88] proposent une méthode pour les RCSF qui s'appelle micro-PKI *(Micro Public Key Infrastructure)*, une version simplifiée des PKI conventionnelles. La station de base possède une clé publique et une autre privée. La clé publique est utilisée par les nœuds du réseau pour authentifier la station de base, et la clé privée est utilisée par la station de base pour déchiffrer les données envoyées par les nœuds. Avant le déploiement, la clé publique de la station de base est stockée dans tous les nœuds. Les auteurs incluent dans leur méthode deux types d'authentification (*HandShake)*. Le premier type d'authentification se fait entre un nœud du réseau et la station de base. Le nœud génère une clé symétrique de session et la chiffre avec la clé publique de la station de base. La clé chiffrée est transmise à la station de base sans être déchiffrée en chemin puisque les nœuds ne connaissent pas la clé privée de la station de base. À la réception, la station de base déchiffre la clé de session et la stocke dans une table. Le deuxième type d'authentification se déroule entre n'importe quel couple de nœuds du réseau en passant par la station de base. Cette dernière joue le rôle de l'authentificateur entre eux. L'un des deux nœuds envoie une requête à la station de base contenant l'identifiant de l'autre nœud. À la réception, la station de base génère une clé aléatoire et la chiffre avec la clé de session correspondante au nœud émetteur de la requête.

Afin de garantir l'intégrité des messages échangés, les auteurs proposent d'intégrer à chaque message un MAC (code) en utilisant la même clé de chiffrement du message. Pour les nouveaux nœuds désirant rejoindre le réseau, il suffit de stocker dans ces nœuds la clé publique de la station de base avant le déploiement.

Watro et al. [89] ont proposé une méthode qui s'appelle TinyPK basée sur l'utilisation des clés publiques et le principe de Diffie-Hellman pour l'établissement d'une clé secrète entre deux nœuds d'un RCSF. TinyPK utilise une autorité de confiance afin de signer les clés publiques des nœuds. La clé de CA est pré-distribuée pour tous les nœuds avant le déploiement afin qu'ils puissent vérifier les clés des voisins après le déploiement. Le choix de l'algorithme RSA pour le chiffrement implique une grande consommation de temps et d'énergie des nœuds. Ainsi, les opérations de base peuvent prendre des dizaines de secondes ce qui va réduire la durée de vie des nœuds du réseau mais aussi impacter sur sa réactivité.

Le protocole symétrique SKKE proposé par *ZigBee* (partie II.3.2.2) repose sur l'utilisation d'une clé maîtresse afin de créer la clé LK (la clé unique partagée entre deux nœuds). Si la clé maîtresse est compromise, la clé LK pourra être compromise aussi. Afin de résoudre ce problème, *ZigBee* a proposé le protocole PKKE [90] basé sur l'utilisation de clés publiques. Comme la création des clés publiques repose sur des données statiques qui ont une courte durée de vie, et non pas sur la confidentialité des clés, la découverte de ces clés ne compromettra pas la clé LK.

La Figure II.28 présente l'établissement de la clé LK en suivant les étapes du protocole PKKE. L'établissement se réalise en quatre étapes. La première consiste à échanger les données statiques (S_u, S_v) accompagnées des données générées par *u* et *v* pour une courte durée (E_u, E_v). S_u et S_v représentent une combinaison contenant les 64 bits des adresses des nœuds (identités des nœuds) avec leur clé publique statique. Les identités des nœuds sont utilisées par les schémas MAC et les clés publiques pour les schémas de partage de clés. E_u et E_v sont des paires de clés publiques générées aléatoirement pour une courte durée. La deuxième étape du protocole consiste à employer une fonction de génération *(KeyBitGen)* qui produit les mêmes clés des deux côtés. Cette fonction est basée sur l'utilisation de la méthode de partage de clé ECMQV *(elliptic curve Menezes-Qu-Vanstone)* spécifiée dans la partie 6.2 de [91]. La troisième étape consiste à dériver les deux clés Mackey et keyData, et à calculer les MAC des deux côtés. Notons que l'octet à ajouter aux paramètres de la fonction MAC_{MacKey} est 02_{16} pour *u* et 03_{16} pour *v*. La dernière étape du protocole consiste

à échanger les MAC calculés afin de les vérifier par *u* et *v* chacun de leur côté. Une fois les MAC vérifiés, la clé partagée sera la clé symétrique KeyData. Notons que la différence entre le protocole asymétrique PKKE et le protocole symétrique SKKE réside dans l'utilisation d'un mécanisme asymétrique (ECMQV) par PKKE afin d'éviter l'utilisation d'une clé maîtresse pré-distribuée et le risque qu'elle soit compromise.

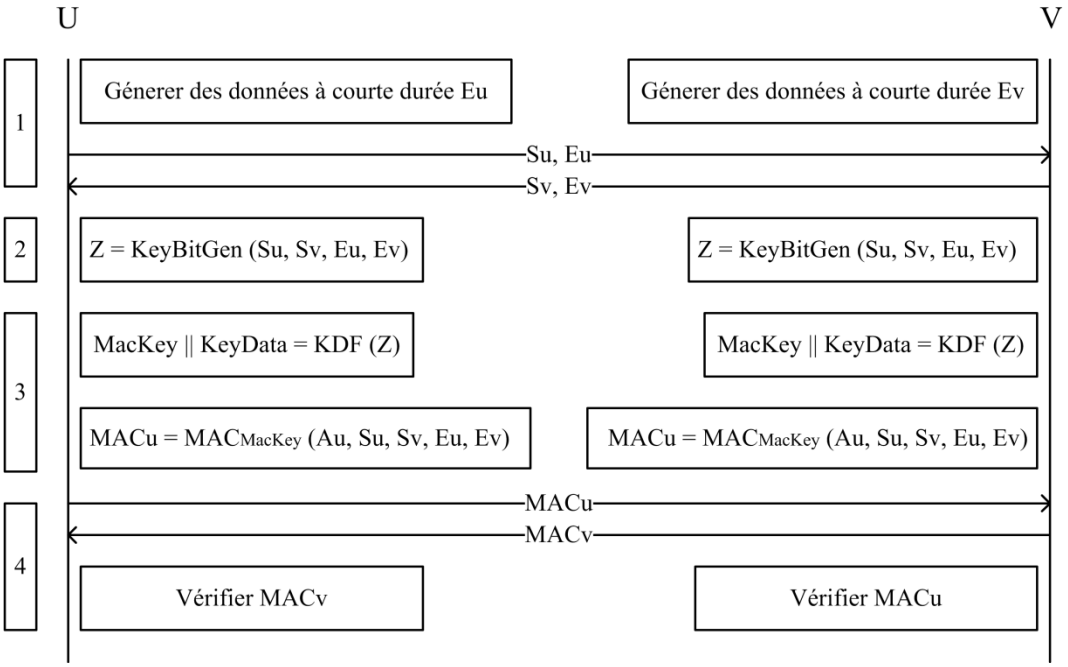


Figure II.28 : Le protocole PKKE de *ZigBee*. Partage d'une clé unique KeyData.

ZigBee a proposé le protocole CBKE [90] *(Certificate-Based Key-Establishment)* afin de mettre en œuvre l'utilisation des certificats digitaux signés par une autorité de confiance. Chaque nœud du réseau possède une clé privée et une clé publique intégrée dans un certificat signé par le CA. Les certificats contiennent des informations spécifiques comme l'ID du réseau, la date de validité, la classe des nœuds, etc. Le protocole CBKE possède une structure identique à celle de PKKE mais ici des certificats ($CERT_u, CERT_v$) sont utilisés à la place des clés publiques (E_u, E_v). La signature du CA est vérifiée dès la réception des certificats par *u* et *v* afin de vérifier l'identité de l'autre nœud.

Lederer et al. [92] ont implémenté une version légère de la méthode d'échange de clés ECDH afin qu'elle soit plus adaptée aux nœuds d'un réseau *ZigBee*. Les auteurs ont combiné plusieurs méthodes qui réduisent le temps de multiplication scalaire des points des courbes elliptiques. Avec une implémentation utilisant une taille de clés de 192 bits, il est démontré

dans [75] que les meilleures performances vis-à-vis d'autres méthodes sont obtenues en utilisant des clés de taille 113 bits.

II.3.3.2 Basées sur l'identité des nœuds

Le but de l'auteur de [66] était d'introduire un nouveau type de méthodes cryptographiques capable d'offrir les mêmes services cryptographiques qu'une PKI sans avoir besoin : (i) d'échanger les clés privées/publiques, (ii) de laisser des répertoires de génération de clés chez l'utilisateur, (iii) ni d'utiliser les services d'une entité tierce de confiance. La méthode a porté le nom IBC *(Identity-Based Cryptography)* sachant qu'elle se base uniquement sur les informations de l'identité de la personne (combinaison de noms, numéro de sécurité social, numéro de téléphone, etc.). L'auteur se base sur l'existence d'un centre capable de délivrer des cartes à puce contenant une clé privée qui correspond aux informations personnelles de la personne qui va la porter. Ainsi, Bob, une personne qui désire envoyer un message à Alice n'a qu'à signer le message avec sa clé privée puis à chiffrer le résultat avec les informations publiques d'Alice avant de l'envoyer. À la réception, Alice déchiffre le message à l'aide de sa clé privée et vérifie la signature de Bob à l'aide des informations personnelles publiques de Bob. L'implémentation du système nécessite d'avoir un schéma à clé publique afin de générer les clés privées. Ce système a été utilisé pour développer de nouveaux systèmes de chiffrement comme le système de cryptographie IBE *(Identity-Based Encryption)* [93].

Les protocoles PKKE et CBKE (partie II.3.3.1) proposés par *ZigBee* utilisent l'identité des nœuds dans leur méthode d'établissement de clés. Le but est d'utiliser ces identités afin de créer une clé partagée unique entre chaque paire de nœuds d'un réseau. Cependant, la création de la clé partagée se réalise avec des interactions entre les deux nœuds. C'est-à-dire, les méthodes exigent l'envoi et la réception de plusieurs messages des deux côtés avant la création de la clé. Afin d'économiser l'énergie des nœuds qui veulent partager un secret et celles des nœuds intermédiaires, plusieurs méthodes ont été proposées afin de supprimer ces interactions. Ces méthodes sont connues dans le domaine de la cryptographie sous le nom de ID-NIKDS [94] *(Identity-Based Non-Interactive Key Distribution Scheme).*

Oliveira et al. [95] ont proposé une méthode qui fait partie des méthodes de ID-NIKDS. Les auteurs utilisent le couplage PBC (voir partie II.2.2.2.9) et le système IBC afin de générer des clés basées sur l'identité des nœuds d'un RCSF. L'idée est que chaque nœud

du réseau possède un identifiant unique et une clé privée. Il est capable de dériver après le déploiement une clé secrète unique avec un autre nœud en connaissant seulement son identifiant.

Jing et al.[96] ont proposé une méthode qui s'appelle C4W basée sur l'utilisation de l'identité des nœuds pour calculer des clés publiques. Ce sont les nœuds eux-mêmes qui sont capables de calculer les clés publiques des autres nœuds en utilisant leurs identités. Ce qui pourrait remplacer le rôle d'un certificat. Avant le déploiement, les nœuds et la station de base sont chargés avec leurs clés propres (clés privées/publiques ECC) et des informations publiques sur les nœuds du réseau. La méthode C4W utilise le principe d'échanges de clés de *Diffie-Hellman* afin de créer une clé unique partagée entre deux nœuds mais sans l'utilisation de certificats.

Jeong et al. [97] ont proposé une méthode hybride d'établissement de clés basée sur le standard ECDH. La méthode utilise des canaux sécurisés pour échanger les données temporaires dédiées à la création des clés uniques partagées entre les nœuds. Et cela dans le but de minimiser les calculs des multiplications scalaires ECC.

II.3.4 Critères de comparaison

Plusieurs critères sont pris en compte afin de comparer les différentes méthodes de gestion des clés. Nous présentons dans la Figure II.29 les critères les plus importants. Nous commençons par le critère de respect de la limitation des ressources des nœuds. La méthode de gestion de clés proposée doit prendre en compte le fait que les nœuds ont été déployés pour collecter les informations. Ils ont besoin de leur espace mémoire pour stocker leurs données et de leur énergie embarquée pour assurer leur rôle applicatif. La solution doit aussi être flexible et dynamique et capable de passer à l'échelle *(scalability).* La gestion des nouveaux arrivants ou le retour des nœuds de la maintenance doivent être gérés d'une façon dynamique. Un autre critère qui doit être respecté est la résilience contre les attaques. En cas de capture de nœuds par exemple, l'adversaire peut utiliser les informations stockées pour mettre en place d'autres attaques et manipuler le réseau. La méthode de gestion de clés doit être capable de détecter les nœuds compromis et authentifier les nœuds du réseau avant de distribuer des clés. Le dernier critère est le renouvellement et la révocation des clés. Nous pouvons le mettre au même niveau d'importance que la distribution de clés. Une clé expirée ou découverte par un adversaire doit être révoquée. Les clés des liens sécurisés doivent aussi être renouvelées

périodiquement. La connectivité d'un réseau est le fait de garantir à ses nœuds d'avoir plusieurs chemins sécurisés pour envoyer ses données. La méthode de distribution de clés doit être capable d'assurer une bonne connectivité du réseau. Le cas d'un départ ou d'une capture d'un nœud peut limiter la connectivité d'autres nœuds avec le réseau. La méthode de distribution doit prendre en compte leur cas en leur proposant des solutions afin de créer de nouveaux chemins sécurisés.

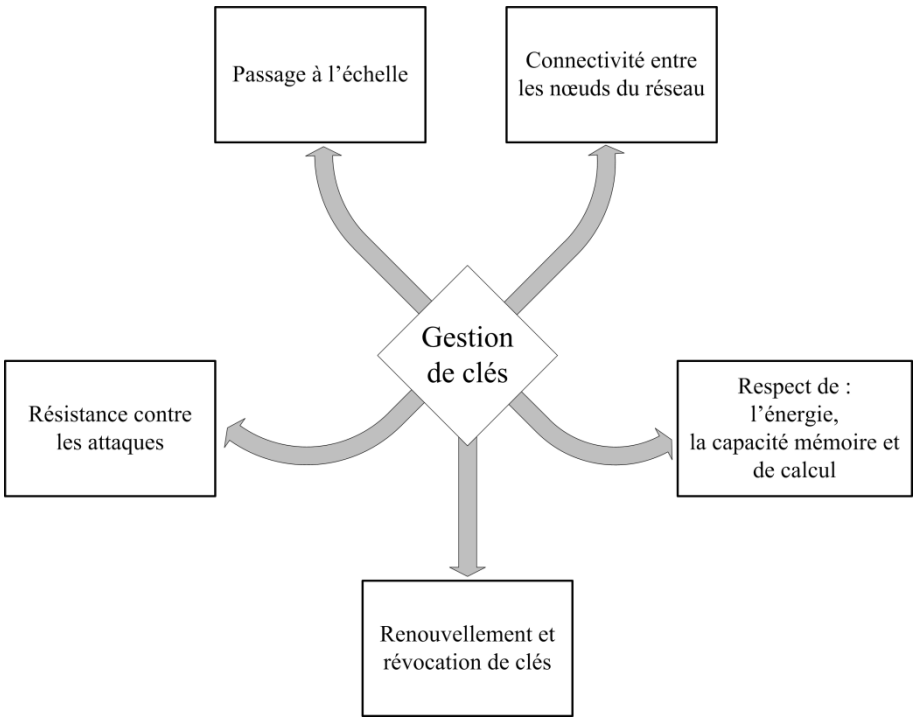


Figure II.29 : Les critères de comparaison des méthodes de gestion de clés des RCSF.

Nous avons étudié différents types de distribution et d'établissement de clés dans les RCSF et nous avons classé les schémas de ces différents types dans la partie II.3.1 (Figure II.20). Dans le Tableau II.5, nous comparons ces schémas en nous basant sur les critères de comparaison de la Figure II.29. Nous avons pris un schéma de chaque catégorie pour alléger le tableau. Notons que le stockage en mémoire évalué dans le tableau prend en considération seulement la taille des clés stockées dans les nœuds et non pas la taille du code des algorithmes et des primitives cryptographiques. Les étoiles du tableau désignent une qualité. Nous avons mis trois, deux ou une seule étoile noire dans la colonne « Connectivité » pour noter que les schémas possèdent respectivement une forte, moyenne ou faible connectivité. Alors que nous avons mis trois, deux ou une étoile dans la colonne « Résistance contre les attaques » pour noter que les schémas possèdent respectivement une forte, moyenne ou faible résistance contre les attaques. En revanche, les ronds du tableau désignent un défaut. Nous

avons mis trois, deux ou un seul rond noir dans la colonne « Coût de ressources utilisées » pour noter que les schémas coûtent respectivement une forte, moyenne ou faible utilisation de ressources.

Le schéma de *Chan et al.*, représentant des schémas probabilistes, montre qu'il consomme peu d'énergie et ne demande pas beaucoup de capacité de calcul. Cependant, les grandes tailles des trousseaux de clés stockés dans la mémoire des nœuds avant le déploiement rendent ce schéma parmi les plus coûteux des schémas symétriques en termes d'occupation de mémoire. Il ne peut pas résister aux attaques telles que les captures physiques des nœuds. Alors que le schéma PIKE assure une meilleure connectivité entre les nœuds du réseau, il passe difficilement à l'échelle.

Schémas			Critères de comparaison								
Type de Schémas	Auteurs des schémas	Basé sur	Passage à l'échelle	Connectivité	Résistance contre les attaques				Coût de ressources utilisées		
					Collection d'informations	Perturbation de communications	Agrégation de données et épuisement de ressources	Capture physique des nœuds	Mémoire (stockage de clés)	Calcul et consommation d'énergie	Renouvellement et révocation
Schémas à clé symétriques	*Chan et al.* [73]	Proba.	Limité	★ ★ ☆	★ ☆ ☆			★ ☆ ☆	● ● ●	● ○ ○	● ● ●
	Chan et Perrig (PIKE) [75]	Déter.	Non	★ ★ ★						● ● ○	
	Perrig et al.(SPINS) [80]	MK + BS	Limité	★ ★ ☆					● ○ ○	● ○ ○	
	Zhu et al.(LEAP) [84]	MK	Limité	★ ★ ☆							
Schémas à clé publiques	*ZigBee (PKKE)* [90]	ID	Juste	★ ☆ ☆	★ ★ ★			★ ★ ☆	● ○ ○	● ● ●	● ● ○
	Oliveira et al. (TinyPBC) [95]	ID + couplage	Juste	★ ★ ★						● ● ○	● ○ ○
	Munivel et al.(micro-PKI) [88]	PKI	Oui	★ ☆ ☆	★ ★ ☆					● ● ●	● ● ○
	Watro et al. (TinyPK) [89]	PKI	Oui	★ ☆ ☆							● ● ●

Tableau II.5 : Comparaison des schémas proposés pour la gestion de clés dans un RCSF. Dans ce tableau le rond est utilisé car nous évaluons un défaut alors que l'étoile est utilisée pour parler d'une qualité.

Les schémas SPINS et LEAP utilisent des clés maîtresses dans l'établissement de clés. Ce qui réduit le stockage de clés dans la mémoire des nœuds. Cependant, la résistance aux attaques est faible. Vu que la clé maîtresse peut être compromise à tout moment, les clés établies après le déploiement en utilisant cette clé peuvent être compromises aussi. En adoptant un système symétrique, ils sont les plus adaptés et parmi les plus rapides en terme de calcul. Notons que les schémas symétriques sont coûteux dans les opérations (si elles existent) de renouvellement et révocation de clés puisqu'ils utilisent des clés secrètes afin d'échanger d'autres clés secrètes. Le problème est plus simple dans les schémas asymétriques puisque les clés publiques n'ont pas besoin d'être secrètes.

Nous pouvons constater que le schéma TinyPBC est le plus convenable des schémas asymétriques. Il résiste à la plupart des attaques connues dans les RCSF. Le fait d'utiliser le couplage afin d'établir une clé unique partagée entre deux nœuds a permis de réduire le besoin de grosse capacité de stockage en mémoire. De plus, la création de cette clé est réalisée sans interaction entre les nœuds ce qui économise le temps de calcul et l'énergie consommée due à ces interactions. Les schémas utilisant le principe des certificats et les PKI restent les plus coûteux en calcul et en consommation d'énergie.

La comparaison entre les schémas symétriques et asymétriques peut différer selon le niveau de sécurité voulu du réseau. Nous remarquons dans le tableau de comparaison que les schémas symétriques peuvent être choisis pour leur rapidité et les schémas asymétriques pour leur résistance contre les attaques.

II.4 Conclusion

Nous avons présenté dans ce chapitre un état de l'art qui détaille les attaques menaçant les réseaux en général et les RCSF en particulier. Pour faire face à ces attaques, nous avons présenté une synthèse bibliographique des systèmes de cryptographie et des mécanismes qui peuvent sécuriser les réseaux filaires. L'adaptation de ces derniers aux RCSF représente de grands challenges. L'absence d'infrastructures comme les PKI dans les RCSF a obligé les nœuds à ne pas faire confiance au réseau et à créer des chemins sécurisés de la source des données jusqu'à la station de base. Des travaux comme [95] ont utilisé les identités des nœuds et le principe de couplage afin de réduire, voire supprimer, les interactions entre les nœuds pour lutter contre un maximum d'attaques. Cependant, jusqu'à ce jour il n'existe pas de solutions complètes et dynamiques facilement adaptables aux RCSF. Toutes les méthodes que

nous avons étudiées dans ce chapitre possèdent de grands avantages lorsqu'elles sont dédiées à des applications et des topologies de réseau particulières. Cependant, il est difficile, voire impossible, de les adapter à d'autres types d'applications ou de les rendre vraiment polyvalentes et universelles.

Chapitre 3

Architecture dynamique pour sécuriser les communications des RCSF

Nous avons étudié dans la partie II.3, les méthodes et protocoles proposés pour les RCSF afin de sécuriser leurs communications. Leur but est, d'une part, d'assurer la plupart des services de sécurité (partie II.1), et d'autre part, de lutter contre un maximum d'attaques tout en respectant les limites matérielles et logicielles des nœuds d'un tel réseau. La comparaison entre les schémas de distribution à clé symétrique et à clé asymétrique de la partie II.3.4, nous conduit à considérer un schéma à clé asymétrique comme solution la plus sécurisée pour l'établissement des clés. En effet, ce schéma garantit une bonne résistance contre la plupart des attaques, et s'accommode de la taille de la mémoire des nœuds en stockant un nombre relativement petit de clés avant le déploiement. Afin de chiffrer les données entre les nœuds, les algorithmes des systèmes symétriques restent les plus rapides et les mieux adaptés à la puissance de calcul des nœuds des RCSF.

Les nœuds du RCSF doivent remédier à l'absence de PKI. Dans notre proposition, la phase de distribution de clés s'appuie sur un RCSF qui exploite la présence d'une station de base (voir Figure I.1). Nous considérons cette station de base comme une entité d'authentification. Ainsi, un nœud désirant rejoindre le réseau passe par une phase d'authentification basée sur l'utilisation de clés asymétriques. Cette phase lui permet de créer un chemin sécurisé vers la station de base, cette dernière peut alors authentifier un nœud et décider d'accepter ou de refuser qu'il fasse partie du réseau. Après avoir été accepté dans le réseau, ce nœud reçoit une clé symétrique pour chiffrer ses données et les envoyer jusqu'à la station de base.

Dans ce chapitre, nous commençons par présenter notre proposition d'architecture. Nous évaluons ensuite notre proposition en termes de résistance contre les attaques en la

comparant à des méthodes existantes. Et nous terminons par une conclusion générale de ce chapitre.

III.1 Proposition d'une nouvelle architecture de sécurité

Notre but étant de proposer une architecture de sécurité complète et dynamique pour les communications des RCSF [98], qui s'adapte à l'évolution du nombre de nœuds. Nous présentons une nouvelle solution s'appuyant sur deux méthodes :

(i) une méthode dynamique de distribution de clés basée sur les systèmes à clé **asymétrique** pour échanger une clé **symétrique** de chiffrement. Cette méthode est composée de trois phases. La première consiste à pré-distribuer à chaque nœud du réseau un ensemble de clés avant le déploiement. La deuxième phase *(JoinNet)* sert à établir des clés symétriques uniques partagées entre deux (ou plusieurs) nœuds du réseau après le déploiement. La troisième phase *(NeighbDisc)* permet enfin à un nœud de découvrir son voisinage dans le but d'établir plusieurs chemins sécurisés avec la station de base. La deuxième phase est réalisée en passant par une phase d'authentification de nœuds via la station de base. Chaque nœud du réseau possède la clé publique de la station de base, qui elle-même possède toutes les clés publiques des nœuds du réseau. Ce partage de clés publiques avant le déploiement permet, après le déploiement, de calculer une clé unique partagée entre chaque nœud et la station de base sans interaction avec elle. D'une part, la clé unique calculée des deux côtés va permettre à la station de base d'authentifier le nœud et ses messages, et d'autre part, lui permettre d'obtenir une clé symétrique qui sera réservée pour le chiffrement de données.

(ii) une méthode complémentaire à la première qui consiste à **renouveler** et **révoquer**, si nécessaire, les clés asymétriques et symétriques distribuées. Cette méthode consiste, d'une part, à renouveler les clés symétriques et asymétriques périodiquement, et d'autre part, à révoquer les clés en possession d'un nœud compromis à l'aide d'un message envoyé par la station de base en *unicast* aux nœuds voisins du nœud compromis. La révocation lance automatiquement la phase de renouvellement de toutes les clés stockées dans les nœuds voisins du nœud compromis.

Dans un premier temps, nous présentons notre méthode dynamique de distribution de clés, nous présenterons notre méthode de renouvellement et de révocation de ces clés dans la partie suivante.

III.1.1 Méthode dynamique de distribution de clés

Nous considérons deux types représentatifs de topologies de RCSF. La topologie plate et la topologie hiérarchique. Puisqu'il existe une différence au niveau des types des nœuds impliqués dans ces deux topologies, nous étudions d'abord les différentes liaisons possibles. Nous présentons ensuite la phase de pré-distribution de clés avant le déploiement, puis la phase d'authentification après le déploiement pour les deux topologies.

III.1.1.1 Etude de liaisons possibles

Dans une topologie plate, il existe seulement deux types de nœuds. Une station de base S qui gère le réseau et des capteurs C_k qui envoient les données récoltées à la station de base. Nous identifions alors deux types de liaisons : (i) $S - C_k$, une liaison possible entre la station de base et un capteur, et (ii) $C_l - C_m$ une liaison possible entre deux capteurs. Dans une topologie hiérarchique de réseau, nous identifions trois types de nœuds : une station de base, des routeurs et des capteurs. Chaque routeur est considéré comme un chef de cluster (ou groupe), il est responsable d'un ensemble de capteurs. Il récolte les données envoyées par les capteurs et les remonte vers la station de base en sollicitant quand c'est nécessaire d'autres routeurs afin de l'atteindre. Nous identifions alors pour cette topologie cinq types de liaisons. Nous retrouvons les types de liaisons déjà cités pour la topologie plate et trois autres types supplémentaires dus à l'existence des routeurs : (i) $S - R_i$, une liaison possible entre la station de base et un routeur, (ii) $R_i - R_j$, une liaison possible entre deux routeurs, et (iii) $R_i - C_k$, une liaison possible entre un routeur et un capteur. Rappelons que les routeurs R_i et les capteurs C_k sont à l'origine des données produites car les routeurs, ainsi que les capteurs, peuvent aussi produire des données.

Dans la Figure III.1 toutes les liaisons possibles entre les nœuds des deux topologies vues précédemment sont réparties en deux classes : (i) les nœuds qui établissent un lien sécurisé direct avec la station de base et, (ii) les nœuds qui établissent un lien sécurisé entre eux en passant par la station de base. Cette répartition se base sur la relation avec la station de base qui joue un rôle important dans la phase d'authentification et établissement de clés dans

notre proposition. C'est la station de base qui décide de l'acceptation des nœuds, de l'établissement des liaisons, et de la délivrance des clés symétriques. Lorsqu'un capteur C_k est à portée de la station de base S et souhaite établir un lien sécurisé avec S, il fera partie des nœuds qui ont un « lien sécurisé direct » avec elle. Après l'authentification du capteur, *S* lui délivre une clé symétrique dédiée au chiffrement de données entre eux. Tandis que si un capteur C_x souhaite établir un lien sécurisé avec un capteur voisin C_y, ils doivent passer par *S*. Le lien entre eux est nommé un « lien sécurisé indirect » par rapport à *S*. La requête est envoyée par l'un des deux capteurs à S. Après l'authentification de la requête et des nœuds, S leur délivre une clé secrète pour le chiffrement.

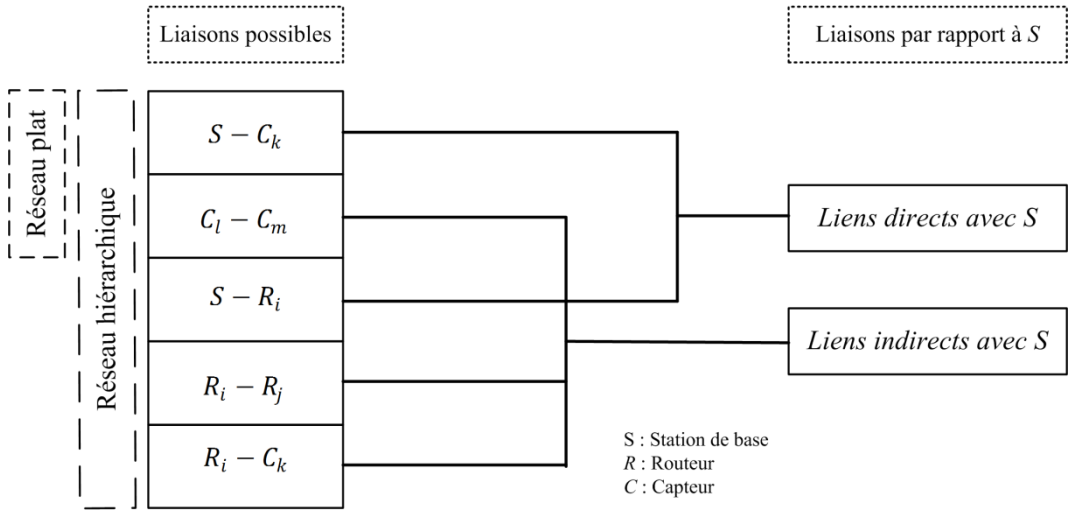


Figure III.1 : Identification des types de liaisons possibles pour établir une clé unique entre deux nœuds.

III.1.1.2 Phase de pré-distribution de clés

La création de nos clés asymétriques est basée sur la cryptographie des courbes elliptiques ECC. Comme nous l'avons vu à la partie II.2.2.2.6, cette cryptographie a déjà fait ses preuves, non seulement elle est la plus rapide des systèmes asymétriques mais aussi elle assure un niveau équivalent, voire plus sécurisé, que les autres systèmes asymétriques [99]. La station de base commence à créer une paire de clés asymétriques unique à chaque nœud du réseau. Par exemple, un nœud A aura une paire de clés unique (P_A, K_A). K_A est choisie aléatoirement dans un intervalle $[1, m]$ où m est un paramètre de la courbe elliptique utilisée. K_A est considérée comme la clé privée de A. P_A, la clé publique de A est obtenue par la multiplication scalaire suivante : $P_A(x_A, y_A) = K_A * G\ (x, y)$, où G est le point à l'infini de la courbe elliptique utilisée.

Un ensemble de clés asymétriques est stocké dans la mémoire des nœuds avant leur déploiement de la façon suivante :

- ***Station de base S :***
  - (P_S, K_S) : la paire de clés ECC unique de *S*.
  - $\{P_i, i \in [1, n]$ où n est le nombre total des nœuds du réseau$\}$: l'ensemble des clés publiques uniques de tous les nœuds du réseau. Ces clés vont servir à l'authentification des nœuds auprès de la station de base.
- ***Un nœud Routeur ou capteur, N :***
  - (P_N, K_N) : paire de clés ECC unique de *N*.
  - P_S : la clé publique de la station de base. Elle va servir à l'authentification de la station de base auprès des nœuds.

III.1.1.3 Phase d'authentification et établissement de clés

La phase d'authentification et d'établissement de clés est une phase initiale qui vient juste après le déploiement. Elle permet aux nœuds de rejoindre le réseau *JoinNet (Join Network).* Cette phase est basée principalement sur la méthode de *Diffie-Hellman* (voir partie II.2.2.2.2). Le but de notre méthode est d'utiliser les clés asymétriques pré-distribuées avant le déploiement afin de partager une clé temporaire. Cette dernière va servir à la station de base à authentifier les nœuds et leur délivrer des clés symétriques uniques utilisables pour le chiffrement des données. Nous commençons alors par expliquer notre adaptation de la méthode de *Diffie-Hellman* puis nous présentons après la phase *JoinNet*.

III.1.1.3.1 Diffie-Hellman sans interaction (DHWI)

Nous utilisons dans notre proposition le mécanisme de *Diffie-Hellman* mais sans interaction entre les nœuds. La pré-distribution des clés asymétriques nous a permis d'éviter les échanges préliminaires de challenges de *Diffie-Hellman*. Nous proposons une version adaptée de ce mécanisme sous le nom DHWI *(Diffie-Hellman Without Interaction)*. L'idée est que chaque nœud du réseau peut établir un lien secret avec la station de base sans interagir avec elle, c'est-à-dire sans aucun échange de messages. Nous illustrons cette méthode par un exemple dans la Figure III.2. Les clés pré-distribuées dans *S* et N_i servent à calculer la clé DH commune sans aucun échange. Cette dernière peut être calculée des deux côtés au même moment ou à des moments différents. Ainsi, cette clé peut être préparée à l'avance (à un

moment d'inactivité par exemple) afin d'optimiser le temps d'attente pour échanger des informations.

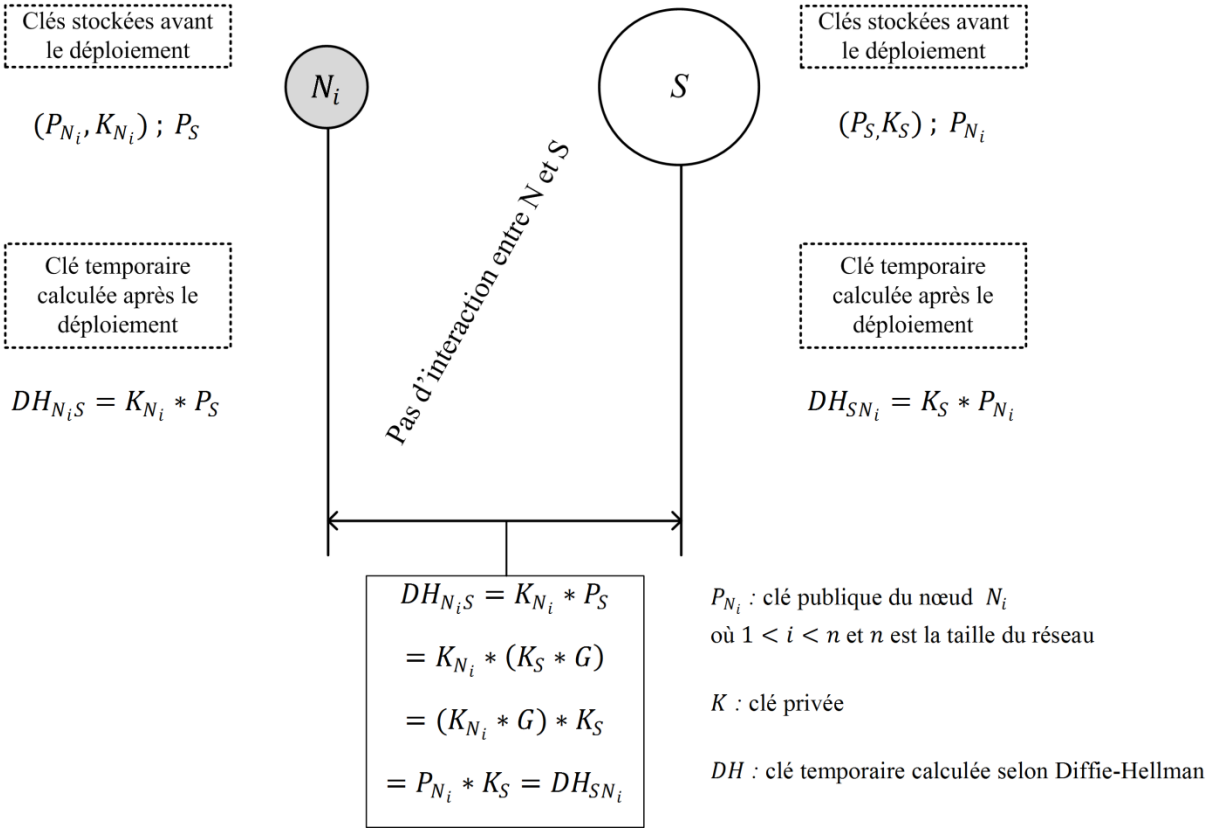


Figure III.2 : DHWI, méthode de partage d'une clé commune selon Diffie-Hellman sans interaction entre deux nœuds.

III.1.1.3.2 La phase d'association au réseau *(JoinNet)*

Nous utilisons le classement de liaisons de la Figure III.1 comme base de notre phase initiale. La station de base *S*, étant le premier nœud du réseau, attend les demandes des nœuds qui souhaitent le rejoindre. Les capteurs (ou routeurs) déployés commencent à envoyer des demandes d'association *(JoinReq)* à *S*. Ces requêtes sont reçues par *S* et traitées selon le type de liens : sécurisé direct ou sécurisé indirect, le traitement de ces deux cas est présenté dans la suite. La station de base ne va pas seulement servir à authentifier les nœuds du réseau, mais elle va aussi pouvoir aider les nœuds à s'authentifier les uns auprès des autres. Notons que l'établissement des clés sécurisées se fait durant la phase *JoinNet*. La station de base reçoit les requêtes, authentifie les nœuds et permet l'obtention des clés symétriques. Ainsi, le lien est construit d'une façon sécurisée dès le début du déploiement et les nœuds qui rejoignent le réseau ne font confiance qu'à la station de base.

III.1.1.3.2.1 Liens sécurisés directs

Lorsque la station de base S reçoit une requête d'un nœud situé à portée, elle vérifie d'abord son contenu. Si la demande d'association concerne le nœud duquel S a reçu la requête, S la considère comme une demande d'établissement de « lien sécurisé direct ». La Figure III.3 (dans la suite p sera la portée d'un lien radio) montre qu'un capteur ou un routeur peuvent envoyer leur(s) requête(s) directement à S pour rejoindre le réseau.

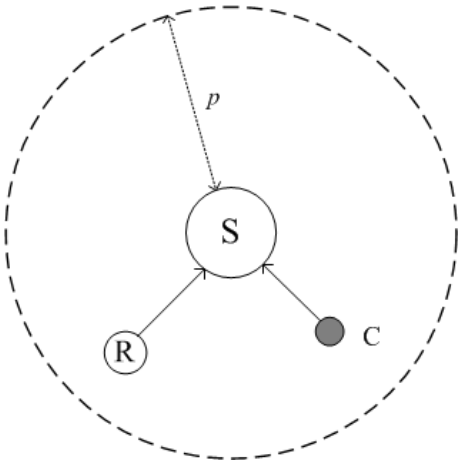


Figure III.3 : Cas d'établissement d'un lien sécurisé.

Les routeurs ou les capteurs de la Figure III.3 génèrent le même type de requêtes. S commence l'authentification des requêtes et des nœuds avant d'accepter qu'ils rejoignent le réseau. Dans la Figure III.4, nous détaillons la phase *JoinNet* d'un nœud désirant rejoindre le réseau d'une façon directe. Nous désignons ce nœud par N_D (nœud établissant un lien direct avec S). Cette phase se termine par l'établissement d'une clé symétrique entre N_D et S. Au début, N_D calcule la clé DH_{N_DS} selon la méthode DHWI en multipliant sa clé privée par la clé publique de S. Les deux nœuds N_D et S sont capables de calculer cette clé car ils sont les seuls à posséder leur propre clé privée. S possède également toutes les clés publiques des nœuds du réseau alors que N_D possède la clé publique de S par pré-distribution avant le déploiement. N_D envoie ensuite la demande *JoinReq* contenant son identité et la date de l'envoi. La requête est reçue et traitée par S après avoir été déchiffrée par la clé calculée DH_{SN_D}. Si S décide d'accepter la demande, une clé symétrique SK_{SN_D} est créée pour être envoyée à N_D. Cette clé est chiffrée à l'aide de l'algorithme asymétrique ECAES (voir partie II.2.2.2.6.3) en utilisant P_{N_D}. Finalement, N_D déchiffre la clé symétrique reçue en utilisant sa clé privée.

La phase *JoinNet* est réalisée en huit étapes comme cela est détaillé Figure III.4. Notons que la première et la troisième étape peuvent être préparées avant le déploiement ou après le déploiement à des moments d'inactivité des nœuds afin de simplifier la phase. La deuxième étape utilise un chiffrement symétrique afin de protéger la confidentialité de la demande envoyée par N_D. L'authentification de N_D auprès de *S* est réalisée dans la cinquième étape. Elle se base sur les informations dans la requête *JoinReq*. La septième étape est basée sur un chiffrement asymétrique afin d'envoyer la clé symétrique créée dans la sixième étape. Finalement, la huitième étape permet à N_D de vérifier l'intégrité et la confidentialité de la clé symétrique en utilisant ECAES.

Notons que *JoinReq* de la Figure III.4 contient l'adresse physique (ID) et l'adresse logique de N_D (affectée selon le protocole de routage après le déploiement) ainsi qu'une valeur aléatoire à usage unique n_D crée par N_D. Cette valeur est connue dans la littérature sous le nom de « *nonces* », une valeur aléatoire à usage unique. Nous la retrouvons aussi sous autres formes, comme celle d'une séquence aléatoire ou d'un challenge utilisé par le protocole SSKE (voir la Figure II.26 de la partie II.3.2.2). À l'étape 4 de la Figure III.4, S extrait l'adresse physique et l'adresse logique de N_D ainsi que la valeur n_D. Ces adresses vont lui permettre d'identifier l'expéditeur de la requête N_D. Alors que la valeur n_D sera intégré dans *JoinRes*, la réponse à la demande d'association de N_D. Ainsi, le nœud N_D pourra s'assurer à l'étape 8 que n_D est la même valeur que celle envoyée à S. L'utilisation des valeurs aléatoires à usage unique à ce stade est nécessaire. Ceci rend le routage imprévisible par l'adversaire. n_D est considéré comme l'identifiant de la requête *JoinReq*. Le fait que N_D a récupéré n_D à la fin de la phase *JoinNet* prouve que la demande a été bien transmis à S.

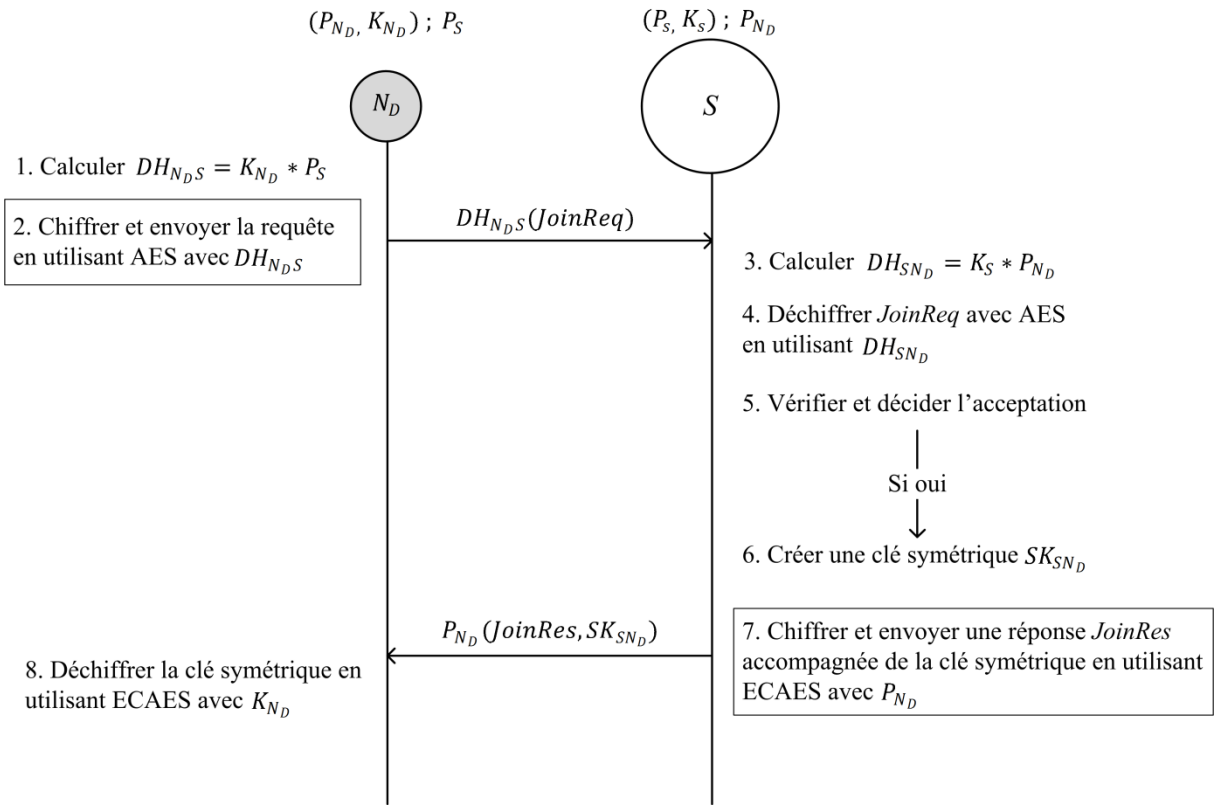


Figure III.4 : Enrôlement d'un nœud à portée direct de la station de base *S* (*JoinNet)*. Huit étapes sont nécessaires pour que ce nœud puisse s'authentifier et établir une clé symétrique avec *S*.

III.1.1.3.2.2 Liens sécurisés indirects

Lorsque la station de base S reçoit une requête d'un nouveau nœud souhaitant rejoindre le réseau à partir d'un nœud existant, elle la considère comme une demande d'un « lien sécurisé indirect ». S extrait du contenu de la requête l'identité du nœud source de la demande. Le nœud existant ne fait que transférer la demande du nœud source. La Figure III.5 montre un capteur et/ou un routeur qui envoient le même type de requêtes à S en passant par des nœuds existants dans le réseau. Notons que les nœuds R_i et C_k peuvent rejoindre le réseau uniquement à partir d'un autre nœud routeur d'une topologie hiérarchique. Rappelons que dans une topologie plate, tout nœud capteur peut servir d'intermédiaire (routeur) pour établir des liens sécurisés indirects.

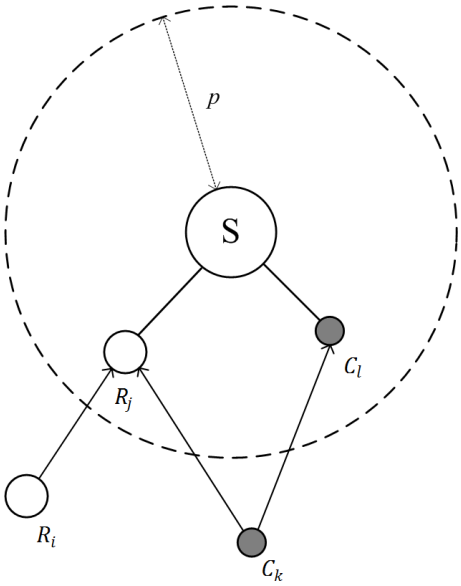


Figure III.5 : Enrôlement d'un nœud qui n'est pas à portée directe de la base. Les nœuds R_i et C_k rejoignent le réseau indirectement.

La station de base S traite les nœuds capteurs ou routeurs indirects de la même manière dans la phase d'authentification avant d'accepter qu'ils rejoignent le réseau. Dans la Figure III.6, nous détaillons la phase *JoinNet* d'un nœud C désirant rejoindre le réseau à partir d'un nœud existant R. Cette phase se termine par l'établissement d'une clé symétrique entre C et R. Au début, C calcule la clé DH_{CS} selon la méthode DHWI en multipliant sa clé privée par la clé publique de S. Ensuite, C envoie la demande *JoinReq* contenant son identité et la date de l'envoi à S via le nœud R. La requête est reçue et traitée par S après avoir été déchiffrée par la clé calculée DH_{SC}. Si S décide d'accepter la demande, elle envoie la clé publique de C à R chiffrée avec la clé DH_{SR}. Notons que cette clé a été déjà calculée et stockée chez S et R lorsque R a rejoint le réseau. R peut déchiffrer P_C à l'aide de sa clé DH_{RS}. Il crée une clé symétrique SK_{RC} pour l'envoyer à C. Cette clé symétrique est chiffrée à l'aide de l'algorithme asymétrique ECAES en utilisant P_C. Enfin, C déchiffre la clé symétrique reçue en utilisant sa clé privée.

La phase *JoinNet* est réalisée en dix étapes. Notons comme précédemment que la première et la troisième étape peuvent être préparées avant le déploiement ou après le déploiement. La deuxième étape utilise un chiffrement symétrique afin d'assurer la confidentialité de la demande envoyée par C. L'authentification de C auprès de S est réalisée dans la cinquième étape. Elle se base sur les informations dans la requête *JoinReq*. La sixième

étape consiste à protéger la clé publique de C avec un chiffrement symétrique. Tandis que la neuvième étape consiste à protéger la clé symétrique créée par R à l'aide d'un chiffrement asymétrique. Finalement, la dixième étape permet à C de vérifier l'intégrité et la confidentialité de la clé symétrique en utilisant ECAES.

Notons que *JoinReq* de la Figure III.6 est similaire à celle de la Figure III.4 du fait qu'elle contient l'adresse physique et l'adresse logique du nœud C ainsi qu'une valeur *nonces* n_C crée par C. À l'étape 7, R extrait la valeur n_C et l'intègre dans sa réponse *JoinRes* envoyée à C. À l'étape 10, C vérifie l'intégrité de la valeur n_C en la déchiffrant avec ECAES.

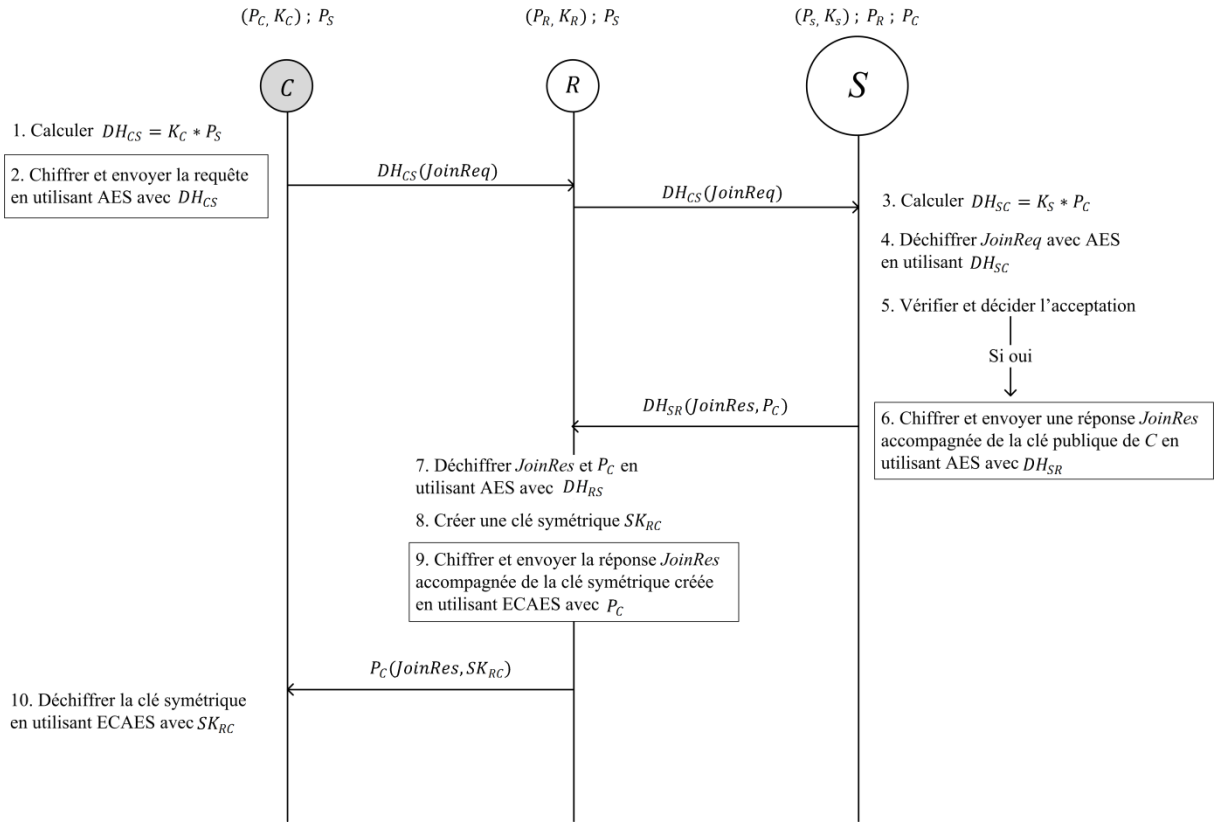


Figure III.6 : Séquencement des actions provoqué par la requête *JoinNet* d'un nœud à lien indirect avec la station de base *S*. Dix étapes sont nécessaires pour que ce nœud puisse s'authentifier auprès de *S* et établir une clé symétrique avec *R*.

III.1.1.3.3 Phase de découverte de voisinage *(NeighbDisc)*

La phase *JoinNet* permet aux nœuds de rejoindre le réseau d'une façon sécurisée. Les nœuds proches de la station de base ont établi des liens sécurisés directs avec elle. Tandis que les nœuds hors portée de *S* ont été obligés de rejoindre le réseau à partir d'un nœud déjà enrôlé dans le réseau. Pour ce faire, le nœud candidat doit obtenir une clé symétrique afin de

pouvoir échanger des données chiffrées avec ce dernier. Cette phase garantit au moins un lien sécurisé de bout-en-bout jusqu'à *S*. Cependant, les nœuds doivent avoir plusieurs liens sécurisés afin de garantir la connectivité des nœuds du réseau. Si un nœud est capturé, les nœuds voisins doivent révoquer le lien sécurisé avec le nœud capturé et découvrir de nouveaux chemins sécurisés allant jusqu'à S. D'où le besoin d'une phase de découverte de voisinage *NeighbDisc (Neighbor Discovery)* permettant à un nœud de découvrir ses voisins et d'établir une clé symétrique unique avec chacun d'eux, dans le cadre d'un *cross-layering* entre la couche application et réseau. Cette phase est réalisée une fois le réseau construit. La Figure III.7 montre un nœud A qui souhaite partager une clé symétrique avec son voisin B. Il chiffre avec DH_{AS} une requête *pkReq* afin d'obtenir la clé publique de B. Une fois la requête reçue, S envoie à A une réponse positive accompagnée de la clé publique de B. Ainsi, A peut déchiffrer la réponse et récupérer P_B. Il crée une clé symétrique SK_{AB}, ajoute au message sa clé publique et le chiffre en utilisant ECAES avec P_B avant de l'envoyer à A. Si SK_{AB} et P_A sont bien déchiffrés par B, ce dernier envoie un *ACK* de confirmation à A chiffré avec SK_{AB}. Cette clé sera dédiée au chiffrement de données entre A et B.

Notons que *JoinReq* utilisée dans la phase *JoinNet* ressemble à *pkReq* de la Figure III.7 du fait qu'elle contient l'adresse physique et l'adresse logique de A et une valeur *nonces* n_A. En revanche, elle contient aussi l'adresse logique de B afin que S puisse envoyer la clé publique P_B de B à A. A retrouve la valeur n_A dans *pkRes*. Selon le même principe, A crée une autre valeur *nonces* n'_A pour l'intégrer dans *skEst.* Puis il va la retrouver en déchiffrant l'ACK reçu de B.

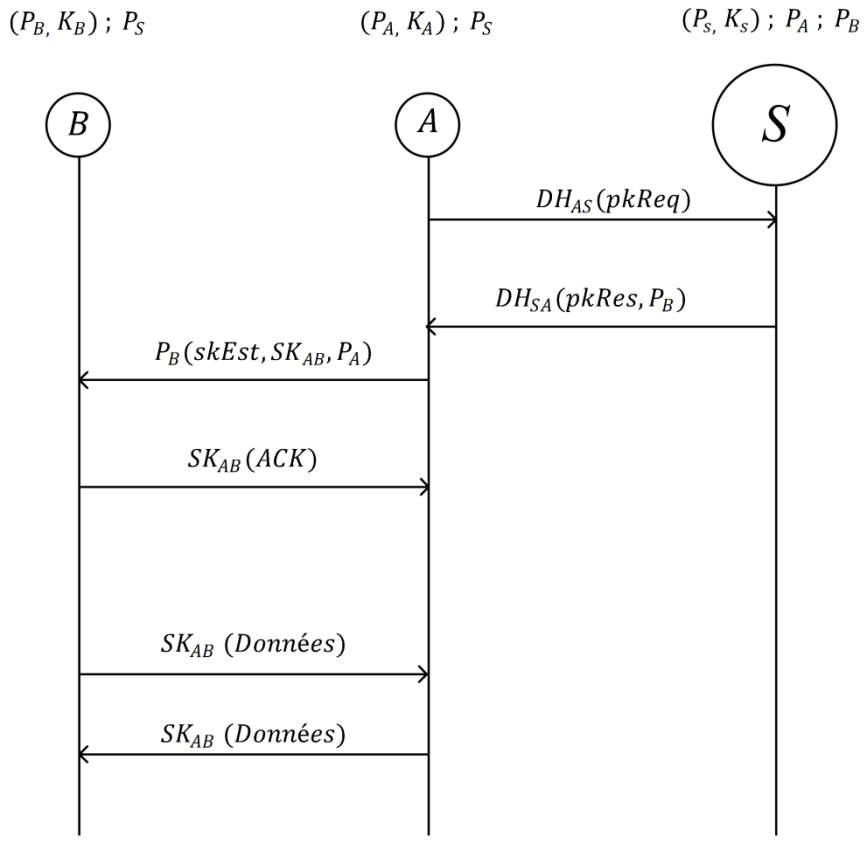


Figure III.7 : Phase de découverte de voisinage *NeighbDisc*. Elle permet à un nœud A d'établir une clé symétrique avec son voisin B.

III.1.2 Méthode de renouvellement et de révocation de clés

Nous commençons cette partie par étudier les clés stockées dans la mémoire des nœuds avant et après le déploiement. Puis nous présentons notre méthode de révocation suivie de notre proposition de renouvellement de ces clés.

III.1.2.1 Etude de clés stockées dans les nœuds

Nous stockons dans chaque nœud avant le déploiement une paire unique de clés asymétriques ainsi que la clé publique de la station de base. Ces clés permettent aux nœuds de calculer une clé symétrique DH partagée uniquement avec la station de base après le déploiement. La clé DH est utilisée pour envoyer des messages de contrôle et des clés publiques. Ces clés publiques permettent d'échanger des clés symétriques créées par les nœuds. Afin de maintenir le niveau sécurité de notre méthode de distribution de clés tout au long de la durée de vie du réseau, nous avons proposé des méthodes pour révoquer ou renouveler les clés distribuées, ou établies, avant et après le déploiement.

Nous allons utiliser la Figure III.8 afin d'expliquer quelles sont les clés stockées dans les différents nœuds du réseau. Dans cette figure nous distinguons : (i) la station de base *S*, (ii) les *i* voisins à un saut de S représentés par le nœud N_x, et (iii) les nœuds du réseau qui ne sont pas à portée de *S*. Pour cette dernière catégorie nous avons choisi de raisonner à partir d'un des voisins à deux sauts de *S* représenté par N_{i+1}. L'extrapolation aux voisins au-delà de 2 sauts de *S* est ensuite possible.

Considérons d'abord, les clés stockées après le déploiement dans la mémoire de ces trois types de nœuds représentatifs. Tous les nœuds du réseau d'un même type stockent leurs clés selon la même stratégie. La station de base S, possède sa propre paire de clés asymétriques et l'ensemble des clés publiques de tous les nœuds attendus dans le réseau. Après le déploiement, S stocke au fur et à mesure une nouvelle clé DH unique (cf. la deuxième colonne de la Figure III.8) chaque fois qu'un nœud rejoint le réseau. Ces clés sont utilisées pour délivrer des clés symétriques pour les nœuds qui sont situés à portée de *S*, et aussi des clés publiques sur demande pour ceux qui sont situés hors de portée de *S*. Une fois la phase d'installation du réseau achevée, un nœud enrôlé pourra alors s'intéresser à son voisinage pour établir plusieurs liens sécurisés. De plus, l'administrateur du réseau pourra mettre à jour localement la table de nœuds attendus de la station de base pour permettre à des nœuds additionnels de rejoindre le réseau, alors qu'ils n'étaient pas déclarés avant le déploiement.

Soit le nœud N_x de la Figure III.8 représentant un des premiers nœuds qui ont rejoint le réseau, puisqu'il est à portée de S, il est considéré comme un nœud direct. Soit le nœud N_{i+1} de cette figure représentant un des nœuds qui ont rejoint le réseau sans être à portée de la station de base. De tels nœuds ont dû établir des liens sécurisés indirects de bout-en-bout avec *S* (en exécutant la phase de *JoinNet*). Pour la Figure III.8 nous avons fait les hypothèses suivantes : (i) N_xa rejoint le réseau avant N_{i+1}, (ii) la requête de N_{i+1} pour rejoindre le réseau a transité par N_x avant d'atteindre *S*. Ceci a permis à N_x d'obtenir la clé $P_{N_{i+1}}$. N_x utilise $P_{N_{i+1}}$ afin de chiffrer $SK_{N_xN_{i+1}}$ et l'envoyer à N_{i+1}. Tandis que N_{i+1} a utilisé sa clé DH afin d'envoyer la demande à S et récupérer une clé symétrique SK de N_x. Notons que les deux nœuds N_x et N_{i+1} ont utilisé leur clé DH afin d'obtenir SK_{NET} envoyée par S et dédiée au chiffrement de la diffusion. Enfin, nous remarquons que la clé DH a joué deux rôles dans notre proposition. Elle a servi à N_x et à N_{i+1} pour obtenir des clés, tandis qu'elle a servi à S pour délivrer des clés.

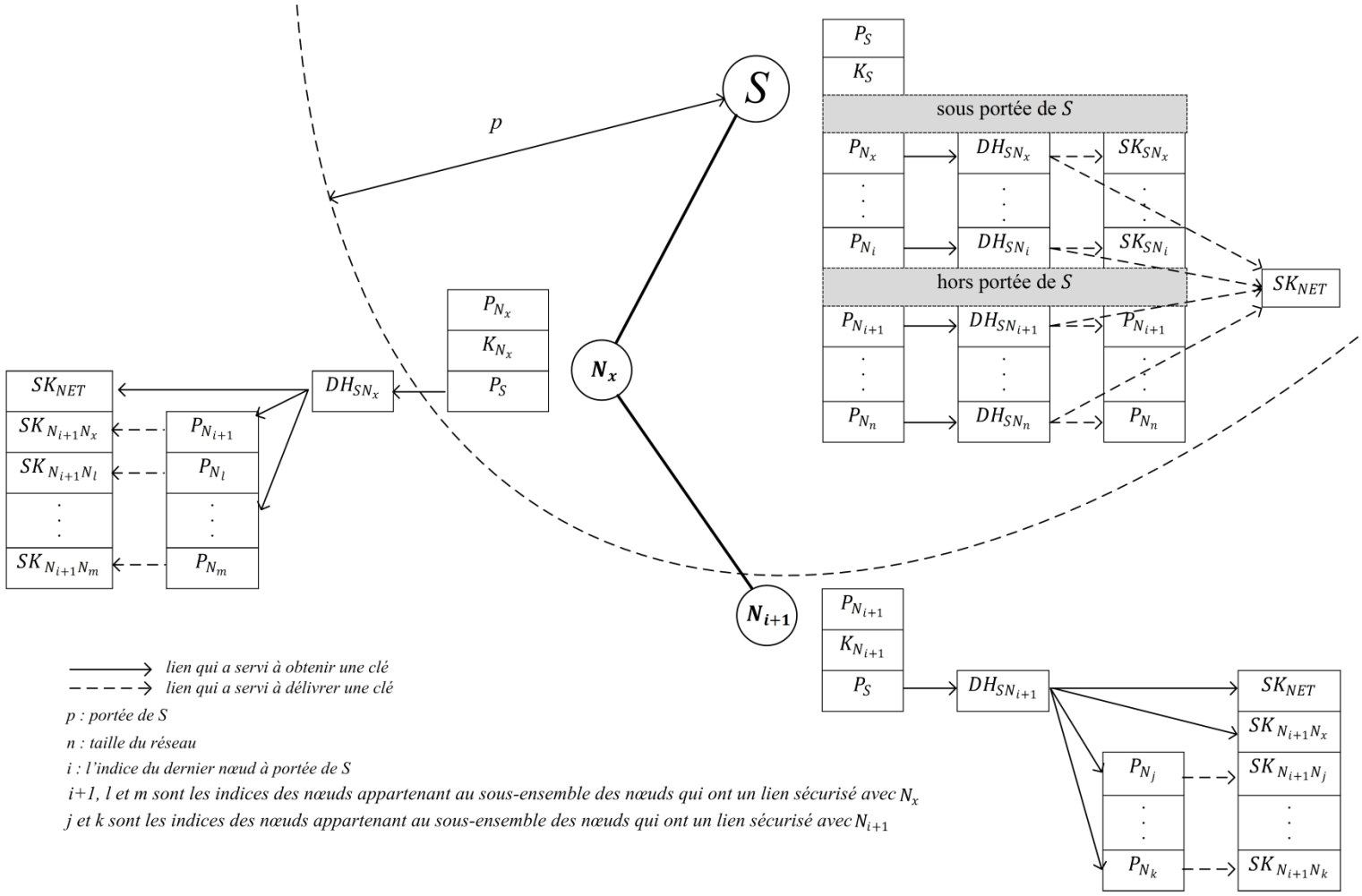


Figure III.8 : Stockage de clés dans la mémoire des nœuds après le déploiement.

D'après la Figure III.8 nous constatons qu'il existe dans le réseau :

- trois types de clés par nœud : (i) une paire de clés propre à lui, (ii) la clé publique de S qui permet de calculer la clé unique DH partagée avec S, et (iii) des clés symétriques uniques propres aux liens sécurisés avec ses voisins et,
- deux types de liens représentatifs : un lien entre la station de base et un nœud à sa portée du réseau, et un lien entre deux nœuds quelconques du réseau.

Notons que la Figure III.8 présente le cas général du stockage de clés des topologies plates et hiérarchiques. Nous pourrions ajouter un nouveau type aux trois types de clés présentés ci-dessus. C'est le cas du partage d'une seule clé symétrique avec plusieurs nœuds du réseau. En effet, un chef de cluster d'une topologie hiérarchique peut créer une clé symétrique unique et la partager avec les nœuds de son cluster. Ceci permettra la diffusion pour tous les nœuds d'un même cluster.

III.1.2.2 La révocation

Comme le montre la Figure III.8, nous avons deux types de liens et trois types de clés à protéger durant toute la vie du réseau. Nos méthodes proposées *JoinNet* et *NeighbDisc* sont suffisamment sécurisées contre la plupart des attaques « logiques » (voir partie III.2) mais elles ne sont pas capables de résister aux effets de la capture d'un nœud. Un nœud peut être capturé physiquement et remplacé par un matériel qui n'a pas de limites de ressources par exemple. Nous proposons une méthode de révocation qui minimise le danger d'un nœud compromis en révoquant toutes les clés et les liens avec ses voisins. Pour décrire cette méthode, nous allons nous appuyer sur la Figure III.9 et nous faisons l'hypothèse que le nœud N_x a été capturé. La révocation se passe en deux étapes :

(i) *la suppression des liens avec* N_x : cette étape consiste à révoquer les liens sécurisés entre N_x et chacun de ses voisins avec qu'il a établi un lien sécurisé. Chacun de ces derniers doit supprimer de sa table la clé publique de N_x et la clé symétrique créée avec lui. Un nœud identifie ses liens sécurisés à partir de sa table. Une fois les clés relatives à un nœud supprimées de la table, le lien avec lui disparaît automatiquement.

(ii) *le lancement d'un mécanisme de renouvellement de clés* : cette étape consiste à renouveler les clés asymétriques de chacun de ses voisins.

Ces deux étapes sont réalisées à la demande de *S*. Une fois que *S* détecte que le nœud N_x est capturé (nous supposons l'existence d'un mécanisme géré par S qui détecte ou soupçonne la capture de nœuds, comme la technique IDS par exemple, voir partie II.1.4.2). S envoie des messages en *unicast* pour atteindre chacun des voisins de N_x. Notons que *S* connaît tous les voisins d'un nœud puisque les phases *JoinNet* et *NeighbDisc* passent par elle. Ces messages sont chiffrés avec la clé DH partagée entre les voisins de N_x avec *S*. Ils contiennent une demande de suppression de liens avec N_x et un lancement du mécanisme de renouvellement de clés asymétriques pour ces nœuds. En effet, N_x possède les clés publiques de ses voisins. Il peut envoyer des requêtes chiffrées avec ces clés publiques pour mettre à jour les clés symétriques. D'où la nécessité de révoquer ces clés. Notons que la clé publique de S n'a pas besoin d'être renouvelée car elle ne présente aucun risque de compromettre d'autres liens ou d'autres nœuds du réseau. Elle permet à N_x de calculer uniquement la clé DH avec S pour lui envoyer des requêtes de contrôle afin d'obtenir des clés publiques des nœuds voisins. Etant donné que S a détecté que N_x a été capturé, elle peut ignorer ses requêtes.

Un nœud quelconque peut ne plus faire partie d'un réseau pour plusieurs raisons. Par exemple, lorsqu'il est récupéré par un administrateur pour une intervention de maintenance, il est détruit suite aux effets d'une cause naturelle, il a fini sa période de d'activité, dans le cas où il a consommé l'intégralité de l'énergie qu'il était censé le faire fonctionner, si les conditions de propagation ont changé au point de modifier la connectivité (présence d'un nouvel obstacle) etc. Lorsque S détecte les effets d'un de ces cas, il lance uniquement la première étape de révocation concernant les voisins du nœud qui a quitté le réseau (suppression des liens sécurisés avec le nœud disparu).

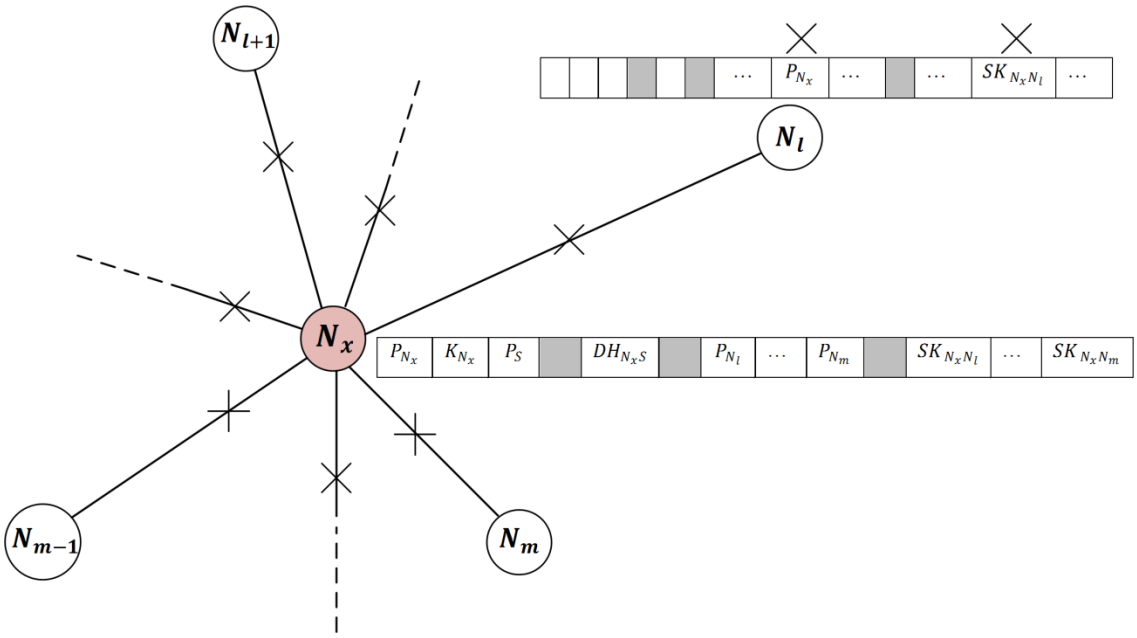


Figure III.9 : Révocation des liens et des clés d'un nœud capturé.

Notre méthode de révocation limite le danger d'un nœud compromis en réalisant les deux étapes présentées précédemment. La phase de *NeighbDisc* va permettre aux voisins du nœud compromis de trouver d'autres liens sécurisés de bout-en-bout vers S. Par conséquent, le fait de compromettre des nœuds du réseau n'empêchera pas le réseau de fonctionner en mode sécurisé, à condition de pouvoir trouver d'autres nœuds à portée.

III.1.2.3 Le renouvellement de clés

Le renouvellement de clés asymétriques et symétriques est nécessaire afin de maintenir le niveau de sécurité de notre méthode. Plusieurs raisons justifient le renouvellement des clés asymétriques et symétriques du réseau :

(i) Renouvellement périodique : cette phase est réalisée après une période T fixée par S. Cette période doit prendre en compte la durée de vie d'une clé utilisée (symétrique ou asymétrique). C'est-à-dire, selon la taille de la clé utilisée, il est possible de donner, pour la technologie que les adversaires sont censés posséder, une estimation du temps nécessaire pour calculer cette clé, et nous fixons la période T inférieure à ce temps.

(ii) Renouvellement à cause d'une capture de nœuds : cette phase est déclenchée par la station de base lorsqu'elle détecte la capture d'un nœud.

(iii) Renouvellement en cas de départ d'un nœud ou d'arrivée d'un nouveau nœud : cette phase concerne uniquement les clusters d'une topologie hiérarchique. Etant donné qu'un chef de cluster R rejoint le réseau avant ses capteurs, il peut délivrer une clé symétrique commune pour tous les membres du groupe. Ainsi, en cas de départ ou d'arrivée de l'un de ses nœuds, R doit renouveler la clé symétrique du groupe pour garantir les propriétés de *forward* et *backward security* (voir partie II.3.2.1.2).

III.1.2.3.1 Renouvellement de clés symétriques

Nous avons proposé dans notre méthode *JoinNet* que le nœud qui existe déjà dans le réseau (A) est celui qui crée et délivre une clé symétrique pour un nouveau nœud B qui rejoint le réseau à partir de A. Ainsi, A est responsable de la création et de la délivrance d'une nouvelle clé symétrique à B. Dans la Figure III.10, nous présentons le mécanisme de renouvellement de cette clé. A crée une nouvelle clé symétrique SK'_{AB} et la chiffre avec la clé commune DH_{AB} (cette clé est déjà calculée des deux côtés au début du déploiement) avant de l'envoyer à B. A la réception, B déchiffre la nouvelle clé et envoie un acquittement de réception en le chiffrant avec SK'_{AB}.

Notons que *JoinReq* utilisée dans la phase *JoinNet* (Figure III.4) ressemble à *skRen* de la Figure III.10 du fait qu'elle contient l'adresse logique de A et une valeur *nonces* n_A. A va retrouver n_A en déchiffrant l'ACK reçu de B.

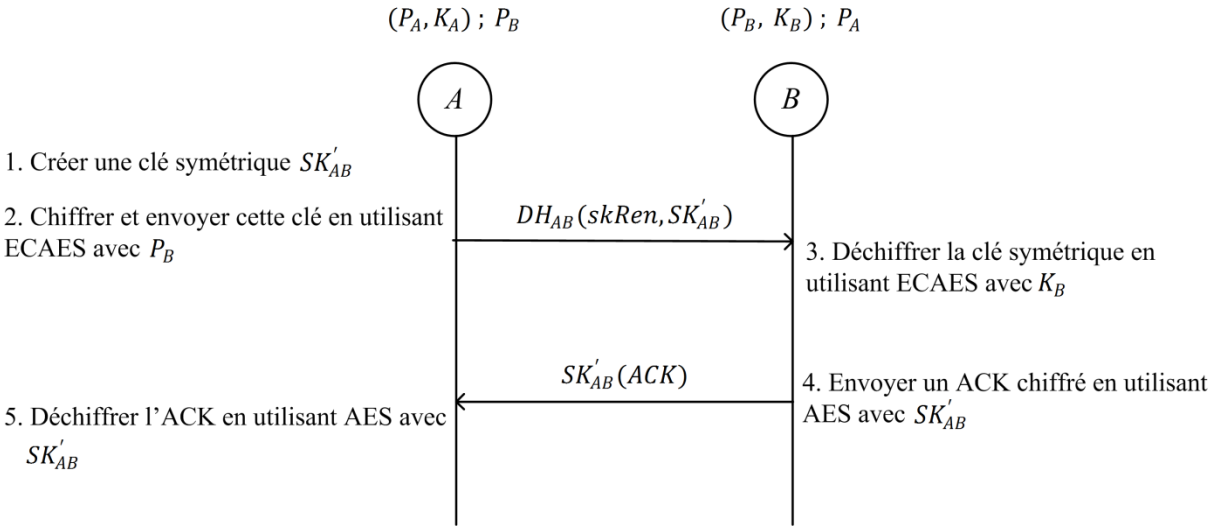


Figure III.10 : Renouvellement d'une clé symétrique.

III.1.2.3.2 Renouvellement de clés asymétriques

Le processus de renouvellent de clés asymétriques est fait par la station de base. Il est lancé pour deux raisons. La première est pour assurer un renouvellement périodique, *S* envoie alors de nouvelles clés asymétriques en *unicast* pour chaque nœud du réseau. Et la deuxième est dans le cas d'une capture de nœuds, *S* envoie alors de nouvelles clés asymétriques en *unicast* aux voisins des nœuds capturés. Dans la Figure III.11, nous présentons le renouvellement d'une paire de clés asymétriques. Elle est créée par *S*, accompagnée de la nouvelle clé publique de *S* et envoyée à un nœud *A* en *unicast*. Dans le renouvellement périodique, *S* exécute le processus de renouvellement pour tout le réseau (sauf pour les nœuds capturés ou partis, etc.) tandis que dans le cas de la capture d'un nœud *B*, *S* crée seulement de nouvelles paires de clés pour les voisins de B. Ainsi, le message sera envoyé sans une nouvelle clé publique de *S*. Notons que les nouvelles clés publiques des voisins de *B* doivent être mises à jour dans la table de leurs voisins. Pour cela, *S* envoie aussi en *unicast* ces clés à ces voisins.

Notons que *pkRen* de la Figure III.11 ressemble à *skRen* de la Figure III.10 du fait qu'elle contient l'adresse logique de S et une valeur *nonces* n_S. S va retrouver n_S en déchiffrant l'ACK reçu de A.

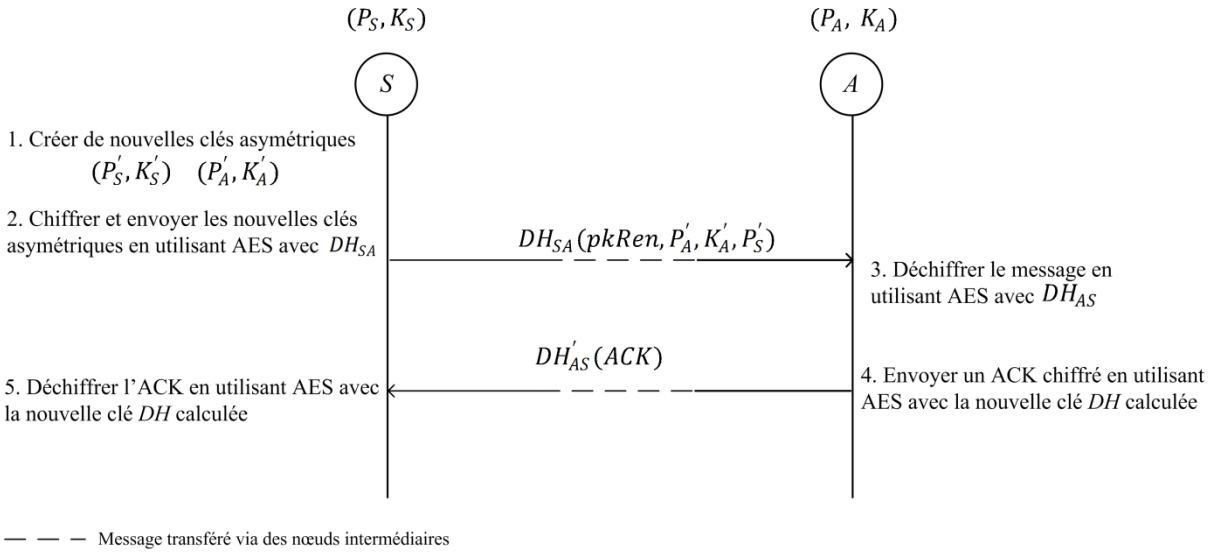


Figure III.11 : Renouvellement de clés asymétriques. Message éventuellement transférés via les nœuds intermédiaires entre S et A.

III.1.3 Application à la topologie de MaCARI

Nous appliquons notre proposition d'architecture sur une topologie hiérarchique afin d'évaluer les différents coûts des méthodes. Nous avons choisi une topologie similaire à celle utilisée dans MaCARI [100], un protocole MAC économe en énergie développé au sein de notre équipe de recherche et des partenaires du projet ANR OCARI [101]. Le protocole se base sur une mise en veille des nœuds du réseau pendant des périodes inactives afin d'économiser de l'énergie. La station de base contrôle la synchronisation de cette mise en veille en fixant des périodes d'activité et des périodes d'inactivité pour éviter les collisions des paquets échangés entre les nœuds.

Dans la Figure III.12, nous montrons la topologie hiérarchique adoptée par MaCARI. La station de base est représentée par une lettre *S*. Les routeurs (aussi appelés coordinateurs dans MaCARI) sont représentés par les lettres *A*, *B*, *C*, *D*, *E*, *F*, et les capteurs par des points de couleur foncée. Nous traitons ici le cas où la station de base est la destination du trafic généré par l'ensemble des nœuds du réseau. Nous supposons que *S* possède une capacité de stockage et de calcul plus grande que celles des autres nœuds. Si nécessaire, nous supposerons que *S* puisse être reliée à un serveur afin de récupérer les données et gérer le réseau. Les capteurs communiquent seulement avec le routeur responsable de leur étoile (architecture du type *cluster tree*). Les données collectées par les capteurs sont envoyées aux routeurs (chefs des étoiles). Ces routeurs, qui sont reliés entre eux d'une façon hiérarchique, renvoient les données reçues de leurs capteurs et de leurs fils, à leur père respectif. Ces données arrivent ainsi naturellement aux routeurs fils de *S*, qui à leur tour, les renvoient vers leur destination, la station de base.

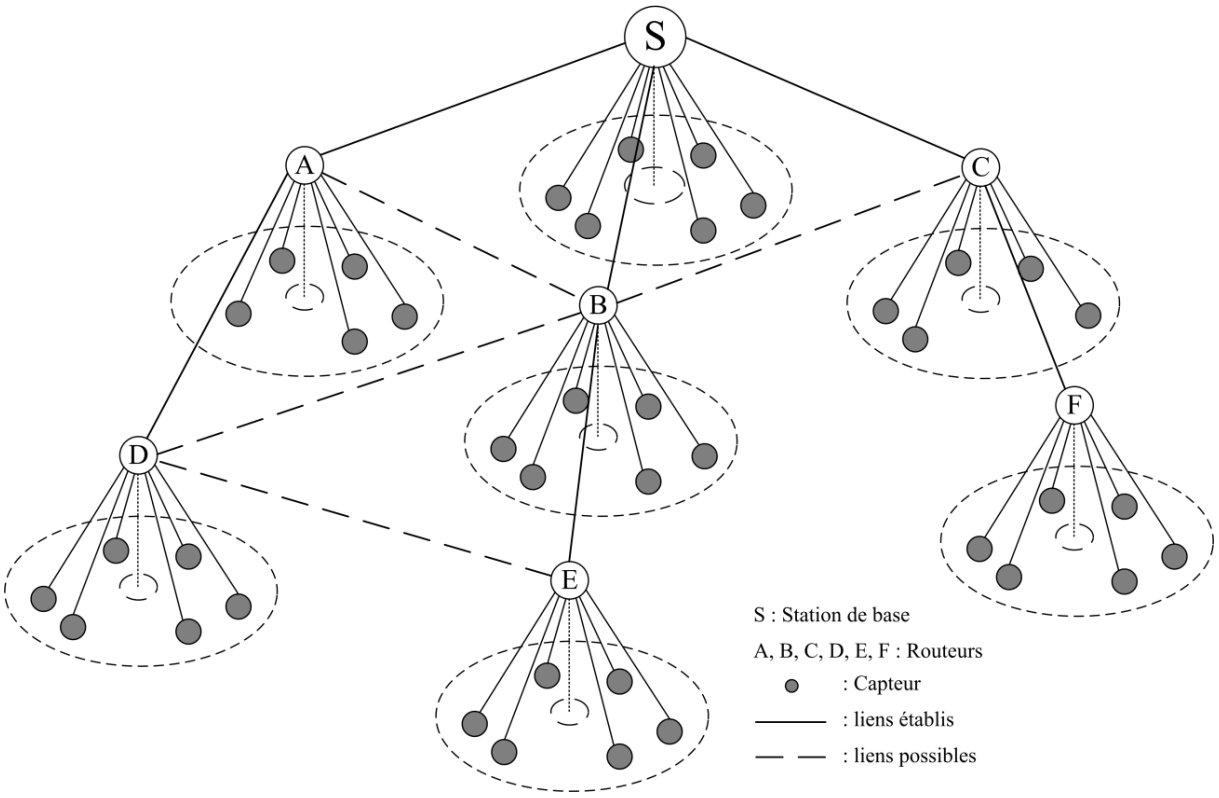


Figure III.12 : La topologie MaCARI d'un RCSF. Le réseau est décomposé en étoiles. Chaque étoile est sous le contrôle d'un routeur (ou la station de base). Les capteurs sont reliés à un routeur et appartiennent à une seule étoile.

En se basant sur la topologie de la Figure III.12, nous soulignons les points suivants :

- la station de base est considérée, d'une part, comme un routeur puisqu'elle est responsable de la gestion d'une étoile et, d'autre part, comme administrateur de tout le réseau en termes de gestion et récupération de données,
- les routeurs sont responsables de la gestion complète de leur étoile. Ils sont considérés comme la seule liaison (un point de passage obligé) entre leurs capteurs et la station de base,
- les capteurs ne communiquent pas entre eux.

Ces observations sur cette topologie nous permettent d'identifier les liaisons suivantes parmi celles définies à partir de la Figure III.1 : (i) $S - C$, une liaison entre la station de base et un capteur de son cluster, (ii) $R - R$, une liaison entre deux chefs de clusters, et (iii) $R - C$, une liaison entre un routeur et un capteur de son cluster. Notons que nous ne retrouvons pas la liaison entre deux capteurs, car dans la topologie adoptée dans le projet OCARI chacun des capteurs confie son trafic au bon soin du routeur auquel il est affilié. Toutes les liaisons

possibles de la topologie sont incluses dans notre étude de liens (voir partie III.1.1.1). Ainsi, notre proposition est applicable à toute topologie de ce type. Nous consacrerons une évaluation détaillée de l'application de notre proposition sur cette topologie dans la partie IV.3.

III.2 Analyse et Discussion

Notre proposition d'une nouvelle architecture de sécurité pour les RCSF présente les avantages suivants :

(i) *utilisation d'une méthode de pré-distribution de clés* : seulement trois clés asymétriques sont stockées dans la mémoire des nœuds du réseau avant le déploiement, auxquelles il faut ajouter uniquement la paire de clés propre à chaque nœud accompagnée de la clé publique de la station de base. Ceci est un grand avantage comparé aux méthodes à systèmes symétriques qui stockent des trousseaux de clés dont la taille peut atteindre celle du réseau.

(ii) *proposition dynamique* : notre proposition est dynamique du fait qu'elle permet aux nouveaux nœuds de rejoindre le réseau à tout moment. Il suffit seulement de stocker les trois clés asymétriques citées précédemment dans le nouveau nœud avant son déploiement, et de le déclarer au niveau de la station de base. Ceci n'est pas pris en compte dans la plupart des méthodes étudiées dans notre état de l'art dans la partie II.3.

(iii) *proposition destinée à deux topologies différentes* : notre architecture est destinée à la fois aux topologies plates et hiérarchiques. La plupart des méthodes de la littérature sont destinées à des topologies spécifiques.

(iv) *utilisation de la station de base comme entité de confiance* : ceci permet à la station de base de remplacer la CA. Elle authentifie les nœuds et décide de les accepter, de les refuser ou de les mettre en attente selon le besoin. Elle est capable d'intervenir et d'envoyer des messages chiffrés en *unicast* à n'importe quel nœud du réseau. Les méthodes qui utilisent ce principe dans la littérature l'ont employé avec des certificats. Ceci est coûteux en termes de signature et vérification.

(v) *utilisation d'un système hybride de chiffrement* : les clés asymétriques pré-distribuées servent à calculer une clé commune entre n'importe quel nœud et la

station de base. Cette clé est utilisée afin d'envoyer des messages de contrôle ou de délivrer de nouvelles clés asymétriques. L'algorithme symétrique que nous avons utilisé est le standard AES. Il utilise les clés symétriques délivrées afin d'assurer avec son chiffrement un maximum de confidentialité. Nous avons utilisé aussi l'algorithme ECAES dans le chiffrement asymétrique afin de délivrer des clés symétriques authentifiées tout en protégeant leur intégrité (MAC (code) inclus dans ECAES).

(vi) *la révocation de clés en cas de capture de nœuds* : notre méthode de révocation permet de limiter le danger causé par une capture de nœuds. Une fois la capture d'un nœud détectée, la station de base intervient en révoquant les clés de ce nœud et en lançant un mécanisme de renouvellement de clés pour des voisins de ce nœud. Seuls les liens entre le nœud capturé et ses voisins sont logiquement coupés. Sauf si le nœud capturé était un point de passage obligé incontournable pour le routage des informations, ceci n'affecte pas le fonctionnement du réseau puisque les nœuds peuvent découvrir d'autres liens grâce à la phase *NeighbDisc*.

(vii) *le renouvellement périodique des clés* : il permet aux nœuds du réseau de renouveler d'une façon sécurisée leurs clés symétriques et asymétriques avant leur expiration. Ceci maintient le niveau de sécurité tout au long de la durée de vie du réseau.

(viii) *utilisation de la méthode DHWI* : elle est plus sécurisée par rapport à la méthode classique DH, et permet d'économiser de l'énergie en supprimant les échanges préliminaires aboutissant à la création de la clé *DH*.

Dans le Tableau II.5, nous avons comparé les méthodes existantes dans la littérature en se basant sur plusieurs critères. Nous pouvons comparer à ce stade notre schéma avec les schémas étudiés dans la littérature en nous basant sur des critères identiques. Nous insérons les caractéristiques de notre schéma au milieu du Tableau III.1. Les étoiles du tableau désignent une qualité. Nous avons mis trois, deux ou une seule étoile noire dans la colonne « Connectivité » pour noter que les schémas possèdent respectivement une forte, moyenne ou faible connectivité. Alors que nous avons mis trois, deux ou une étoile dans la colonne « Résistance contre les attaques » pour noter que les schémas possèdent respectivement une forte, moyenne ou faible résistance contre les attaques. Nous remarquons que notre schéma présente la résistance la plus efficace face aux effets produits par les captures de nœud. Notre

schéma est équivalent à *micro-PKI* et *TinyPK* en termes de passage à l'échelle. Cette équivalence vient du fait que ces deux schémas utilisent des clés asymétriques permettant de passer à l'échelle plus facilement. Nous remarquons qu'en termes de connectivité, notre schéma est équivalent à plusieurs schémas symétriques, cela à cause de l'utilisation de la phase *NeighbDisc* après le déploiement. N'importe quel nœud peut établir un lien avec son voisin en demandant sa clé publique à la station de base. Nous soulignons que notre comparaison est partielle car elle ne tient pas compte des autres critères du Tableau II.5, comme l'utilisation des ressources. Nous compléterons notre comparaison dans le chapitre 4 (voir partie IV.3.4) par une évaluation de notre schéma et de schémas existants en termes de coûts de renouvellement et révocation de clés et de ressources utilisées.

Schémas		Critères de comparaison				
Type de Schémas	Auteurs des schémas	Basé sur	Passage à l'échelle	Connectivité	Résistance contre les attaques	
					Collection d'informations, Perturbation de communications Et Agrégation de données et épuisement de ressources	Capture physique de nœuds
Schémas à clé symétriques	*Chan et al.* [73]	Proba.	Limité	★ ★ ☆	★ ☆ ☆	★ ☆ ☆
	Chan et Perrig (PIKE) [75]	Déter.	Non	★ ★ ★		
	Perrig et al.(SPINS) [80]	MK + BS	Limité	★ ★ ☆		
	Zhu et al.(LEAP) [84]	MK	Limité	★ ★ ☆		
Notre proposition		**BS**	**Oui**	★ ★ ★	★ ★ ★	★ ★ ★
Schémas à clé publiques	*ZigBee (PKKE)* [90]	ID	Juste	★ ☆ ☆	★ ★ ★	★ ★ ☆
	Oliveira et al. (TinyPBC) [95]	ID + couplage	Juste	★ ★ ★		
	Munivel et al. (micro-PKI) [88]	PKI	Oui	★ ☆ ☆	★ ★ ☆	
	Watro et al. (TinyPK) [89]	PKI	Oui	★ ☆ ☆		

Tableau III.1 : Comparaison de notre proposition avec des méthodes existantes en termes de connectivité, passage à l'échelle et résistance aux attaques. Dans ce tableau l'étoile est utilisée pour parler d'une qualité.

III.3 Conclusion

En utilisant les techniques actuelles de sécurité et en adaptant des mécanismes comme DHWI, nous avons proposé une nouvelle architecture de sécurité dédiée à deux types de topologies pour les RCSF. Notre méthode *JoinNet* permet aux nœuds de rejoindre à tout

moment le réseau. Ceci rend notre proposition dynamique et lui donne l'avantage par rapport aux méthodes proposées dans la littérature. Une fois le réseau déployé, la station de base authentifie tous les nœuds et délivre des clés symétriques dédiées au chiffrement de données. Même après le déploiement, l'administrateur du réseau est capable de retirer des nœuds du réseau pour les maintenir et les déployer de nouveau. Notre méthode présente une bonne résistance aux effets produits par la capture des nœuds puisque seuls les liens entre un nœud capturé et ses voisins sont touchés. Les nœuds sont capables de découvrir de nouveaux liens avec leurs voisins en utilisant la méthode *NeighbDisc* pour proposer une nouvelle opportunité de routage sécurisé de l'information sur le réseau. La comparaison effectuée dans la partie III.2 nous permet de conclure que notre méthode possède un niveau de sécurité supérieur contre les attaques comparée à ceux décrits dans notre synthèse bibliographique. Il nous reste néanmoins à étudier à quel(s) coût(s) tous les mécanismes décrits peuvent être implémentés avec la technologie de RCSF d'aujourd'hui. Ceci fera l'objet de notre quatrième chapitre.

Chapitre 4

Implémentations et résultats

Nous présentons dans ce chapitre une implémentation de notre proposition d'architecture sécurisée ainsi que les principaux résultats qui la caractérisent. Ces résultats sont des mesures réelles qui permettent d'évaluer le coût de chaque phase de notre proposition. Dans la première partie de ce chapitre, nous présentons le matériel utilisé pour notre implémentation. Dans la seconde partie, nous abordons tout d'abord l'implémentation des algorithmes utilisés puis nous évaluons les ressources qu'elle nécessite en termes d'occupation mémoire, de temps d'exécution et de consommation d'énergie. Avant de faire le choix de l'algorithme symétrique de notre implémentation, nous avons implémenté, évalué, et comparé deux algorithmes symétriques concurrents. Pour le choix et l'implémentation des algorithmes asymétriques, nous nous sommes basés sur la bibliothèque TinyECC [102]. Nous avons choisi et adapté des algorithmes de cette bibliothèque afin qu'ils répondent à nos besoins. Dans la troisième partie de ce chapitre, nous présentons l'évaluation de notre proposition en termes : (i) d'empreinte mémoire avant et après le déploiement, (ii) de temps d'exécution et de consommation d'énergie, ceci pour les phases *JoinNet* et *NeighbDisc*, le renouvellement et la révocation de clés, (iii) d'estimation de l'effet de la taille de réseau. Ensuite, nous comparons notre méthode de distribution de clés avec des méthodes existantes en termes d'utilisation de ressources. Et nous terminons ce chapitre par une conclusion générale.

IV.1 Matériel utilisé pour l'implémentation

Nous commençons cette partie par la présentation de TinyOS [103], un système d'exploitation typiquement utilisé pour programmer des cartes dédiées à la recherche et à l'évaluation de solutions de RCSF. Ensuite nous présentons les propriétés logicielles et matérielles des cartes TelosB [104] qui ont servi pour implémenter et évaluer notre proposition.

IV.1.1 Système d'exploitation TinyOS

TinyOS est un système d'exploitation conçu pour promouvoir les RCSF (il ne nécessite aucune licence). Il a été développé à l'origine par l'université de Berkeley [105] de Californie (États-Unis), à la fin des années 1990. Ce système d'exploitation a acquis rapidement un succès international [106] en devenant une solution de référence pour le maquettage et l'évaluation de RCSF. Il est programmé en NesC, un dérivé du langage C. NesC propose une architecture basée sur des composants permettant de réduire considérablement l'empreinte mémoire du système et de ses applications. Le fonctionnement de TinyOS s'appuie sur la gestion d'évènements. Ainsi, l'activation de tâches, leur interruption ou encore la mise en veille du capteur s'effectuent suite à l'apparition d'évènements. L'implémentation de composants s'effectue en déclarant des tâches, des commandes ou des évènements : (i) une tâche est un travail de longue durée (à l'échelle du temps processeur), (ii) une commande est l'exécution d'une fonctionnalité précise dans un autre composant, et (iii) un événement est l'équivalent logiciel d'une interruption matérielle.

IV.1.2 Cartes TelosB

Nous citons les deux familles de cartes, les plus utilisées de nos jours, dédiées à la recherche et à l'évaluation de solutions dans le domaine des RCSF: (i) la famille « Mica », composée de trois types de cartes : MicaZ [107], Mica2 et Mica2dot, et (ii) la famille « Telos », composée de deux types de cartes TelosA et TelosB [104]. Les deux familles ont été produites par « Crossbow » [108]. Les cartes Mica ont été proposées à l'utilisation par *Hill et al.* en 2001-2002 [109] tandis que les cartes Telos ont été introduites par *Polastre et al.* [110] (université de Berkeley) en 2004-2005. Nous avons implémenté notre proposition sur les cartes TelosB. Dans la Figure IV.1, nous présentons une carte TelosB à côté d'une pièce de monnaie afin d'estimer la taille de la carte. Une carte TeloB comprend :

(i) Un port USB via lequel la carte peut se brancher à un ordinateur par exemple. Son utilisation ne demande pas l'installation de logiciel spécifique, la carte apparaît comme un port série.
(ii) Un bouton de réinitialisation qui permet de réinitialiser la carte sans avoir besoin de débrancher son port USB.
(iii) Un bouton utilisateur dédié à des besoins spécifiés dans le programme de l'utilisateur. Il est possible de demander l'exécution d'une commande en appuyant

sur ce bouton (envoyer des informations d'affichage via le port série à l'ordinateur connecté, par exemple).

(iv) Trois voyants pilotés lorsque l'utilisateur le demande dans son code (utilisés pour le diagnostic et la mise au point). Par exemple, l'activité du voyant rouge peut être programmée pour qu'il s'allume lorsqu'une tâche se déclenche ou se termine.

(v) Un microcontrôleur MSP430 cadencé à 8 MHz qui dispose de 10 KB de mémoire vive (RAM) et de 48 KB de mémoire non volatile (ROM).

(vi) Un module radio CC2420 travaillant sur une bande de fréquences radio s'étendant de 2,4 à 2,4835 GHz qui décharge le processeur de certaines tâches pour l'accès au médium, l'échange de trames, etc.

(vii) Une antenne permettant à la carte d'envoyer ou de recevoir le signal radio.

(viii) Deux piles situées de l'autre côté de la carte. La tension des piles peut varier de 2,1 à 3,6 Volts en courant continu. Lorsque la carte est connectée via le port USB, elle en reçoit 5 Volts. Nous ferons l'hypothèse comme dans [111] (page 4) que 3 Volts suffisent.

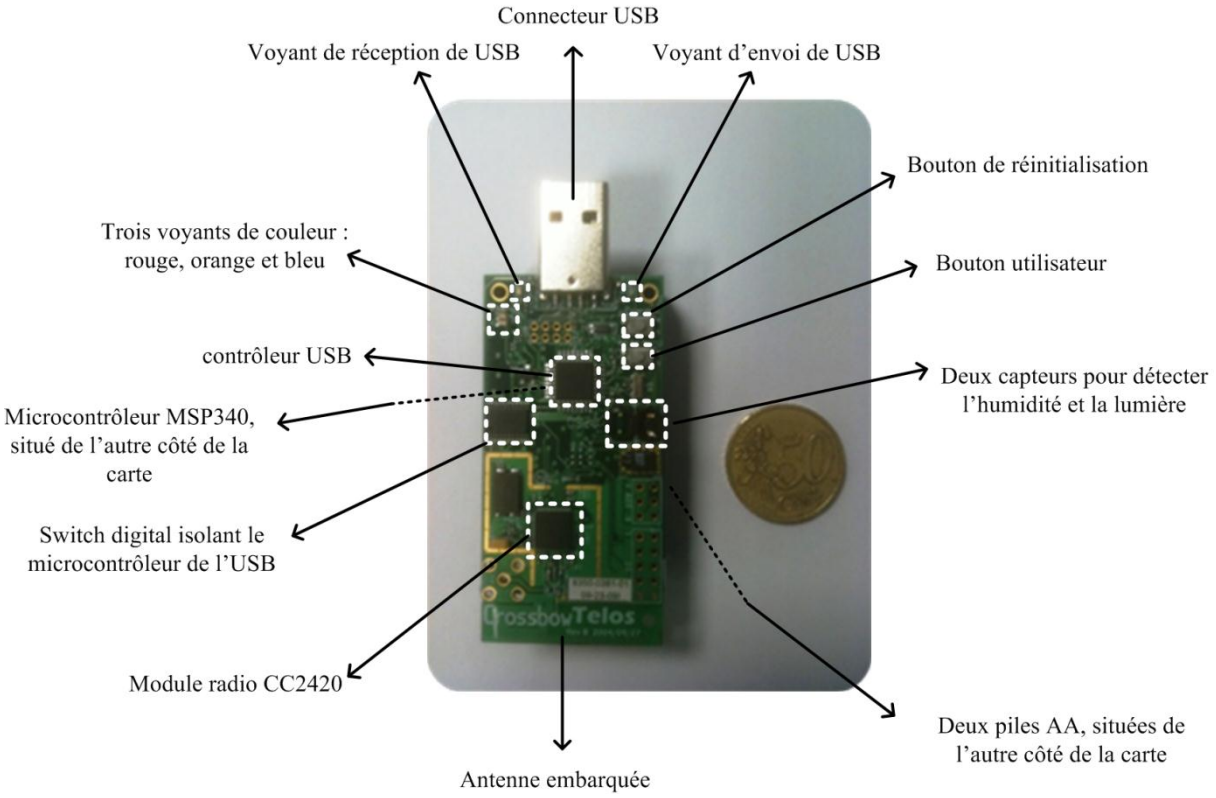


Figure IV.1 : Une carte TelosB.

IV.2 Implémentation d'algorithmes sur TelosB

Dans les RCSF, la durée de vie des nœuds et la vie du réseau en général représentent une contrainte importante. La complexité des algorithmes utilisés pour le chiffrement doit respecter la capacité d'exécution du micro-processeur des nœuds. Avant d'évaluer notre proposition en termes d'empreinte mémoire, temps de calcul et consommation d'énergie, nous présentons d'abord nos implémentations d'algorithmes symétriques et asymétriques [112].

IV.2.1 Algorithmes symétriques

Avant de faire le choix de l'algorithme symétrique à retenir pour notre proposition, nous en avons implémenté, évalué, et comparé [113] deux : (i) l'algorithme AES en mode compteur [114] (AES-CTR) utilisé par plusieurs standards comme ZigBee [115], WirlessHART [6] et ISA100.11a [7], (ii) et l'algorithme du chaos proposé par les auteurs de [47]. Nous avons présenté dans la partie II.2.2.1.2 le fonctionnement de ces deux algorithmes. Nous présentons dans les parties suivantes l'implémentation de leurs principales fonctions ainsi que leur évaluation.

IV.2.1.1 AES-CTR

L'algorithme AES-CTR ressemble en quelque sorte à un flux de chiffrement *(stream cipher)*. En effet, un vecteur contenant un compteur est généré, chiffré puis appliqué aux données avant leur transmission. Nous avons implémenté AES-CTR afin de chiffrer les données entre deux nœuds d'un RCSF. Les messages sont divisés en blocs de 16 octets. Ces blocs sont chiffrés en utilisant une clé de 128 bits et cela en dix tours *(rounds).*

Une première phase est nécessaire afin de créer les éléments basiques de l'algorithme AES-CTR. Dans l'Algorithme IV.1, nous présentons le pseudo-code de cette phase. La clé symétrique d'entrée est stockée à l'aide de la fonction *KeyInit()*. Ensuite, la dérivation de cette clé est créée à l'aide de la fonction *KeyExpansion(Key)* et est stockée dans une matrice de dimension 44×4 octets. Cette matrice est divisée en 11 blocs durant le processus du chiffrement. Enfin, un vecteur contenant un compteur initial $Count_0$ de 16 octets est créé en utilisant la fonction *CountGenInit()*. Ce compteur est incrémenté avant chaque transmission.

Notons que la fonction *SboxGen()* génère une matrice *Sbox* de 16×16 octets qui sera utilisée ultérieurement durant le chiffrement.

Algorithme 1 : fonction d'initialisation d'AES-CTR : ***InitAES***

1. *Key[16]* ← *KeyInit()*
2. *KeyExp[44][4]* ← *KeyExpansion(Key)*
3. *Sbox[16][16]* ← *SboxGen()*
4. $Count_0[16]$ ← *CountGenInit()*

Retourner : *KeyExp, Sbox,* $Count_0$

Algorithme IV.1 : La fonction d'initialisation d'AES-CTR.

La fonction de chiffrement/déchiffrement d'AES-CTR est présentée par le pseudo-algorithme dans l'Algorithme IV.2. Notons que la fonction de déchiffrement d'AES-CTR est exactement la même que celle de chiffrement. Cette fonction prend en entrée 128 octets de données par défaut. L'essentiel est que cette entrée soit un multiple de 16. Dans le cas où nous avons besoin de chiffrer un bloc de taille inférieure à 16 octets, nous complétons ce bloc par un bourrage de zéros. Le champ existant dans l'entête de chaque message envoyé indique la taille des données chiffrées. Ainsi, lors du déchiffrement d'un message, le nœud récepteur peut enlever le bourrage ajouté et extraire les données de base en clair. La fonction *InitAES*() est appelée afin d'initialiser les matrices *KeyExp, Sbox* et le vecteur $Count_0$. La boucle *Pour* est là afin d'appliquer le chiffrement à tous les blocs de 16 octets. Pour chacun de ces blocs, la fonction *CounterCipherGen* est appelée afin de créer un vecteur de flux de chiffrement/déchiffrement *CountCipher*. L'opération XOR permet d'appliquer le chiffrement/déchiffrement aux blocs en clair afin d'être stockés dans *OutputBytes.* Notons que l'opération «*i* : *i+16* » retourne les vecteurs compris entre *i* et *i+16.*

Algorithme 2 : fonction de chiffrement/déchiffrement d'AES-CTR : ***EncryptAES***

Entrée : *InputBytes[128]*

1. *(KeyExp, Sbox,* $Count_0$*)* ← *InitAES()*
2. *CountCipher[16]* ← $Count_0$ *[16]*
3. **Pour** *i=0* → *128 ;* **pas de** *16 ;* **faire**
   i. *CountCipher* ← *CounterCipherGen(KeyExp, Sbox, CountCipher)*
   ii. *OutputBytes[i : i + 16]* ← *InputBytes[i : i + 16]* ⊕ *CountCipher*
4. **Fin pour**

Retourner : *OutputBytes[128]*

Algorithme IV.2 : La fonction de chiffrement/déchiffrement d'AES-CTR.

La fonction *CounterCipherGen* est détaillée dans l'Algorithme IV.3. Le vecteur de compteur *Count* est stocké dans une matrice de dimension 4×4 octets appelé *State*. Le premier bloc de 4×4 octets de la matrice *KeyExp* est stocké dans une matrice appelée *W. W* est utilisée dans la fonction *AddRoundKey* afin d'être ajoutée à la matrice *State*. Ensuite, la boucle *Pour* intervient afin d'assurer un chiffrement/déchiffrement de 9 tours *(rounds)* appliqué à la matrice *State*. Ce chiffrement est réalisé par : une substitution (*SubBytes*), un changement de ligne (*ShiftRows*), un mixage de colonnes (*MixColumns*), et une addition d'une partie de la dérivation de clés (*AddRoundKey*). Le dixième tour est fait en appliquant à la matrice *State* une dernière substitution, un dernier changement de ligne et une dernière addition du dernier bloc de *KeyExp*.

Algorithme 3 : Génération d'un vecteur de flux chiffré d'AES-CTR : ***CounterCipherGen***

Entrée : *KeyExp[44][4], Sbox[16][16], Count[16]*

1. *State[4][4]* ← *Count[16]*
2. *W[4][4]* ← *KeyExp[0 :3][4]*
3. *AddRoundKey(State, W)*
4. **Pour** *tour=0* → *9 ;* **pas de** *1 ;* **faire**
   i. *SubBytes(State, Sbox)*
   ii. *ShiftRows(State)*
   iii. *MixColumns(State)*
   iv. *W* ←*KeyExp[(tour × 4) : (4 + tour × 4)-1][4]*
   v. *AddRoundKey(State, W)*
5. **Fin pour**
6. *SubBytes(State, Sbox)*
7. *ShiftRows(State)*
8. *W* ← *KeyExp[40 :43][4]*
9. *AddRoundKey(State, W)*
10. *CountCipher[16]* ← *State*

Retourner : *CountCipher[16]*

Algorithme IV.3 : La fonction de génération d'un flux chiffré d'AES-CTR.

IV.2.1.2 Algorithme du chaos

Les auteurs de [47] ont proposé une méthode qui consiste à mixer et perturber deux sorties de la fonction du chaos PWLCM avant d'appliquer le chiffrement aux données en clair. Nous avons présenté les avantages de cette méthode dans la partie II.2.2.1.2.2. Le pseudo-code de la principale fonction de cet algorithme est présenté dans Algorithme IV.4. Cette fonction prend en entrée 128 octets de données par défaut. L'essentiel est que cette entrée soit un multiple de 2 octets vu que la fonction du chaos fabrique en sortie deux valeurs stockées sur deux octets. L'opération de chiffrement/déchiffrement comprend deux fonctions majeures : PWLCM pour la génération de deux valeurs pseudo-aléatoires comprises entre 0 et 1, et LFSR utilisée régulièrement[3] afin de perturber les valeurs aléatoires obtenues précédemment pour obtenir deux nombres perturbés à la sortie. La boucle *Pour* parcourt le

[3] La fonction de perturbation est appliquée chaque Δ itérations.

vecteur *InputBytes* avec un pas de 2 valeurs. Pour chaque 2 octets de données en clair, deux valeurs pseudo-aléatoires sont générées et perturbées (R_1, R_2). Notons que la bibliothèque de NesC n'offre pas la déclaration de variables en double précision. Par conséquent, R_1 et R_2 sont des nombres décimaux avec seulement 6 chiffres après la virgules. Ils sont multipliés par 10^6 et stockés sous la forme de deux valeurs entières NB_1 et NB_2. Finalement, le chiffrement/déchiffrement est fait en effectuant l'opération XOR (octet par octet) entre, d'une part, deux octets de données en clair, et d'autre part, le résultat sur deux octets d'un XOR entre NB_1 et NB_2. Notons que le symbole « % » signifie le reste de la division et Δ est un nombre d'itérations, paramètre fixé au début de la méthode afin d'appliquer la perturbation régulièrement.

Algorithme 4 : La fonction de chiffrement/déchiffrement de l'algorithme du chaos : ***EncryptChaos***

Entrée : *InputBytes[128]*

1. $X_0 \leftarrow 0{,}950347$, $Y_0 \leftarrow 0{,}567217$
2. $p_1 \leftarrow 0{,}372134$, $p_2 \leftarrow 0{,}292134$
3. $\Delta \leftarrow 30$, $Q_1 \leftarrow Q_2 \leftarrow 32768$
4. *NbIter* $\leftarrow 0$
5. **Pour** $i=0 \rightarrow 128$; **pas de** 2 ; **faire**
   - i. $X_i \leftarrow PWLCM\ (X_i, p_1)$
   - ii. $Y_i \leftarrow PWLCM\ (Y_i, p_2)$
   - iii. *NbIter* $\leftarrow$ *NbIter* + 1
   - iv. ***Si*** *NbIter* % $\Delta=0$ ***alors***
     - a. $R_1 \leftarrow LSFR\ (X_i, Q_1)$
     - b. $R_2 \leftarrow LSFR\ (Y_i, Q_2)$
   - v. ***Fin Si***
6. $NB_1[2] \leftarrow R_1$
7. $NB_2[2] \leftarrow R_2$
8. $OutputBytes[i : i + 1] \leftarrow InputBytes[i : i + 1] \oplus (NB_1 \oplus NB_2)$
9. **Fin pour**

Retourner : *OutputBytes[128]*

Algorithme IV.4 : L'algorithme du chaos de [47].

Notons que l'initialisation de la génération de clés est composée de deux phases : (i) l'initialisation des deux valeurs (X_0, Y_0) et leurs paramètres (p_1, p_2) présents dans la fonction PWLCM (nous avons fixé ces valeurs aléatoirement), et (ii) l'initialisation du degré L de LSFR des deux perturbations appliquées aux valeurs (X_0, Y_0). Si N est la précision finie (en bits) correspondant à la taille d'une valeur du processeur utilisé, la clé générée aura $2^{2(2N-1)+2L}$ combinaisons possibles. Dans notre implémentation sur les cartes TelosB, nous pourrons fixer N avec un maximum de 16 bits. En effet, la bibliothèque de base utilisée (NesC) avec ce microcontrôleur n'offre pas une double précision décimale (*float precision*). Nous avons initialisé aussi L à 16 bits afin de perturber tous les bits générés. Dans ce cas, l'ensemble des valeurs possibles pour les clés appelé espace clé a une taille de 2^{94}. Ceci nous permet de chiffrer tous les échanges pendant toute la durée de vie du réseau sans avoir besoin de changer les valeurs initiales (X_0, Y_0) et leurs paramètres (p_1, p_2) [116]. De plus, une petite modification de (X_0, Y_0) entraîne un changement radical des valeurs de sortie de la carte du chaos vue la sensibilité des cartes du chaos aux conditions initiales [47].

IV.2.1.3 Évaluation et comparaison

Nous avons évalué l'algorithme AES-CTR et l'algorithme du chaos en termes de temps d'exécution et de qualité de chiffrement en utilisant la démarche suivante. Dans un premier temps, nous avons flashé puis exécuté le code des deux algorithmes sur les cartes TelosB afin d'obtenir des mesures réelles. Ensuite, ce code est transformé en utilisant Matlab afin d'exécuter le code sur un ordinateur avec un processeur Pentium à double cœur T4200 avec une vitesse allant jusqu'à 2 GHz. Cette démarche a été effectuée dans le but d'analyser le temps nécessaire aux opérations de chiffrement des deux algorithmes exécutées sur une machine ordinaire et de le comparer à celui obtenu quand ces opérations sont exécutées sur les cartes.

IV.2.1.3.1 Comparaison du temps d'exécution

Dans cette partie, nous nous concentrons sur le temps d'exécution des deux algorithmes. Nous avons mesuré le temps consommé par les deux algorithmes pour chiffrer 128 octets de données. Comme le montre le Tableau IV.1, pour les cartes TelosB, le temps nécessaire pour chiffrer ou déchiffrer 128 octets en utilisant l'algorithme du chaos est 844% plus rapide que celui d'AES-CTR. Tandis que sur l'ordinateur le ratio est seulement de 286%. Ceci est dû à la différence de puissance de calcul des deux processeurs. Notons que le temps

d'exécution de la phase d'initialisation des paramètres de l'algorithme du chaos est négligeable. En revanche, la fonction d'initialisation d'AES-CTR (voir Algorithme IV.1) prend 1,92 secondes sur l'ordinateur et en moyenne 12, 29 secondes sur les cartes TelosB. Nous avons constaté que la fonction *SboxGen()* consomme en moyenne 12, 28 secondes. Nous avons réduit le temps global de cette phase en pré-calculant la matrice *Sbox* et en la stockant dans la mémoire des cartes avant le début du chiffrement. *Sbox* n'utilise pas d'informations extraites de la clé de chiffrement. C'est pourquoi nous pouvons la pré-calculer et la stocker dans les nœuds du réseau avant leur déploiement. Ainsi, le temps d'initialisation d'AES-CTR est réduit à 8,8 ms.

	Nombre d'octets chiffrés/déchiffrés	AES-CTR	Algorithme du chaos	Ratios
Ordinateur	128	132,32	46,25	286%
	1	1,03	0,36	
TelosB	128	1631,4	193,2	844%
	1	12,74	1,5	

Tableau IV.1 : Temps d'exécution en millisecondes de l'AES-CTR et l'algorithme du chaos.

Le Tableau IV.2 montre le temps nécessaire en millisecondes pour exécuter les deux fonctions principales de l'algorithme du chaos PWLCM et LFSR. Notons que la fonction LFSR dédiée à la perturbation des valeurs prend plus de temps à s'exécuter que la génération de ces valeurs. Cette perturbation régulière est pourtant une étape nécessaire afin d'éviter les boucles de la fonction de PWLCM. Nous avons choisi d'appliquer la perturbation chaque 30 itérations aux valeurs de sortie de PWLCM. Nous avons constaté que cette valeur est suffisante pour les faire sortir des cycles périodiques (répétition de valeurs de sortie).

	PWLCM (N=16 bits)	LFSR (L=16 bits et $\Delta = 30$)
Ordinateur	0,031	0,71
TelosB	0,75	1,83

Tableau IV.2 : Temps d'exécution en millisecondes des deux principales fonctions de l'algorithme du chaos.

IV.2.1.3.2 Ressemblance d'images

La démarche en trois temps, que nous avons choisie pour évaluer la qualité de chiffrement, consiste à étudier son impact de son application sur une image comme proposé par les auteurs de [117], [118] et [119]. Nous avons, pour cela, chiffré une image de référence en utilisant les deux algorithmes et nous avons calculé la ressemblance entre l'image d'entrée en clair et les images chiffrées obtenues. Nous avons utilisé la célèbre image de Lenna. Pour des raisons de simplification, nous avons choisi une version noir et blanc de cette image. En effet, Le chiffrement d'une image en couleur demande le chiffrement de trois images (R, G et B) puis de les fusionner.

IV.2.1.3.2.1 Résultats de chiffrement

Nous avons appliqué le chiffrement des deux algorithmes à une image de dimension 128×128 pixels de Lenna. Chaque pixel est codé sur 1 octet. Ainsi, la taille totale de l'image est de 16 384 octets. Les cartes TelosB utilisent des trames basées sur le format de trame de la norme IEEE 802.15.4 limité à 133 octets [120] par trame. Pour cela, l'image doit donc être découpée en plusieurs blocs de moins de 133 octets de long afin qu'elle soit chiffrée et transmise. Nous avons divisé l'image en vecteurs (blocs correspondant à une portion horizontale de l'image) de 100 octets transmis dans des trames de 133 octets.

La Figure IV.2 montre l'image en clair en (a), l'image chiffrée en utilisant AES-CTR en (b) et l'image chiffrée en utilisant l'algorithme du chaos en (c). Notre première constatation est que le résultat de ces deux chiffrements montre que les deux algorithmes ont réussi à bien cacher l'image initiale.

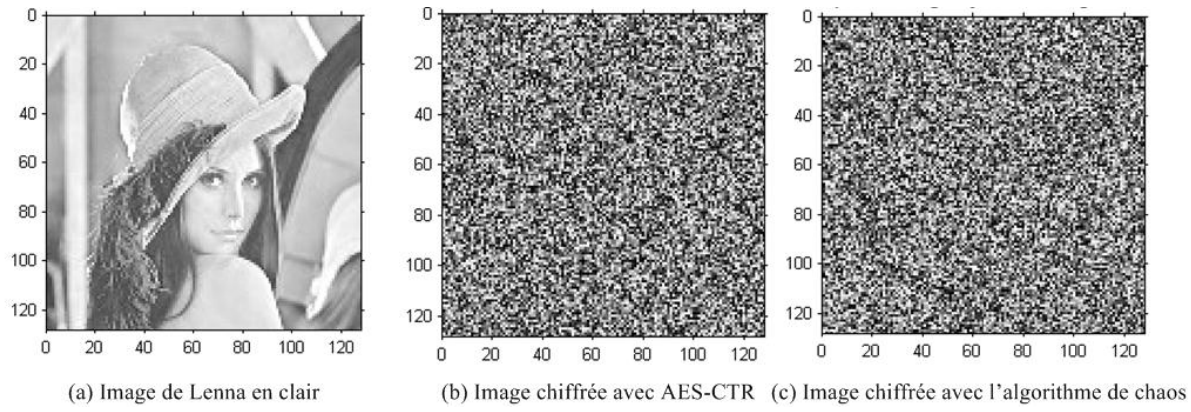


Figure IV.2 : Le résultat de l'image de Lenna chiffrée avec AES-CTR et l'algorithme du chaos.

Afin d'analyser les effets du chiffrement nous avons utilisé la distribution des niveaux de gris sur les trois images qui viennent d'être introduites dans la Figure IV.2. La Figure IV.3 nous donne la distribution des niveaux de gris pour les trois cas. L'histogramme en (a) montre que la figure de Lenna a une distribution très typée des niveaux de gris. Tandis que les histogrammes des deux images chiffrées avec les deux algorithmes montrent une distribution uniforme. Cela nous mène à conclure que les deux algorithmes ont chiffré l'image d'une façon quasi-équivalente.

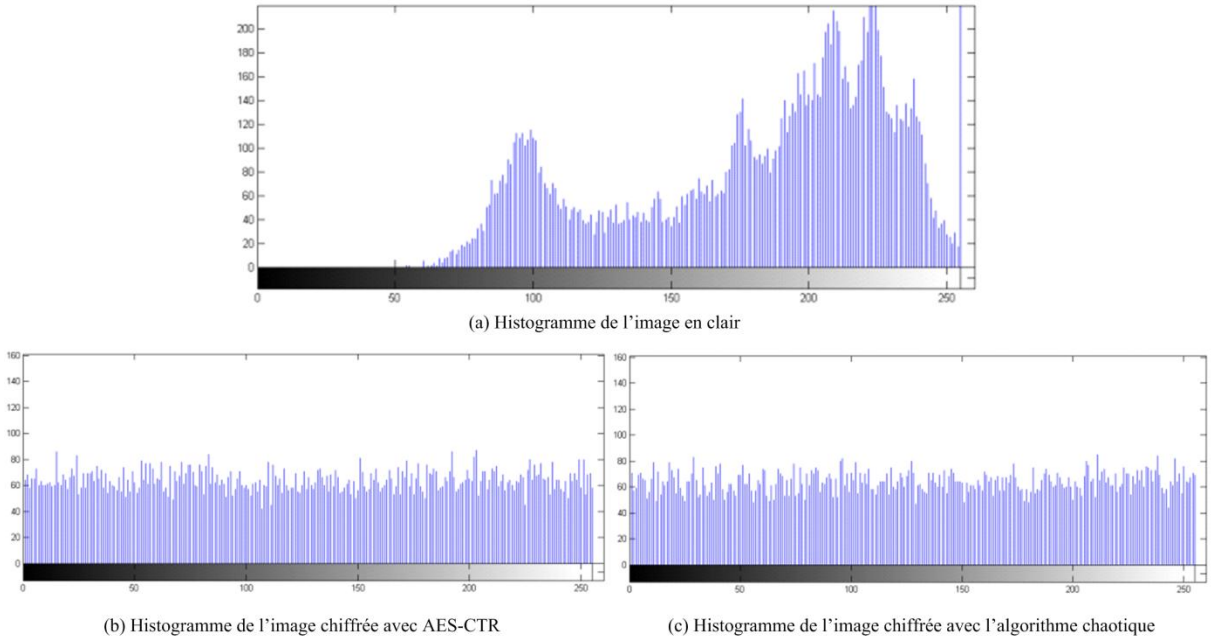
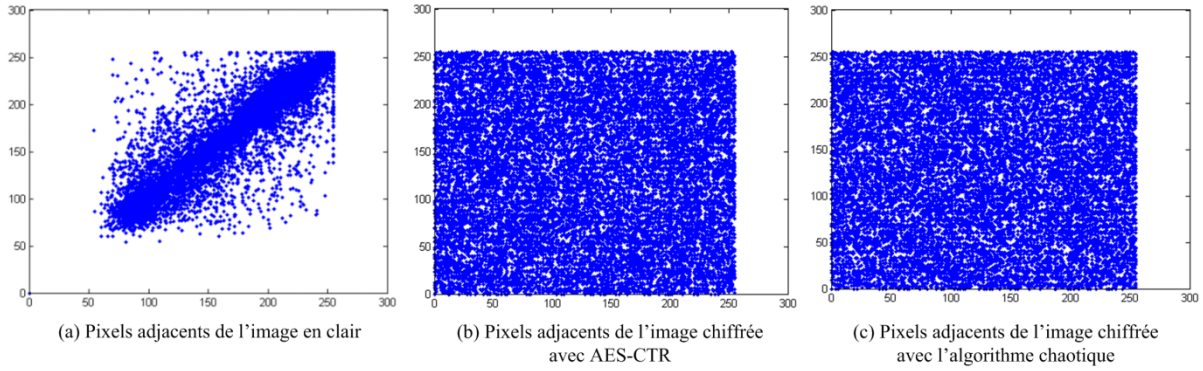


Figure IV.3 : La distribution des niveaux de gris *(grayscale colorbar)*.

IV.2.1.3.2.2 Coefficient de corrélation

Le deuxième temps de notre démarche consiste à exploiter un facteur de corrélation entre les pixels de l'image comme autre méthode pour évaluer la qualité de chiffrement. Concrètement il s'agit de quantifier l'existence de zones de mêmes couleurs ou de même densité de niveau de gris dans notre cas. Pour chaque image chiffrée, il s'agit de mesurer le facteur de corrélation entre chaque pixel et ses pixels adjacents (chaque point sur l'axe des x présente une valeur d'un pixel qui varie entre 0 et 255 et nous trouvons sur l'axe des y la valeur de son pixel adjacent). Dans l'image en clair, partie (a), la corrélation entre les pixels adjacents est clairement élevée. Tandis que dans les images chiffrées (b) et (c), la corrélation est très faible. Nous pouvons conclure que le chiffrement de deux valeurs consécutives avec

les deux algorithmes a donné deux valeurs complètement différentes et assez loin des valeurs d'entrée.



(a) Pixels adjacents de l'image en clair

(b) Pixels adjacents de l'image chiffrée avec AES-CTR

(c) Pixels adjacents de l'image chiffrée avec l'algorithme chaotique

Figure IV.4 : La corrélation entre les pixels adjacents de l'image de Lenna en clair et chiffrée avec AES-CTR et l'algorithme du chaos.

Le coefficient de corrélation est un facteur important qui montre la corrélation entre deux pixels adjacents selon trois axes : horizontalement, verticalement ou diagonalement. Il s'agit de la mesure de l'étendue et de la direction de combinaisons linéaires de deux pixels adjacents. Si deux pixels ont deux valeurs proches, la corrélation est proche des deux extrémités de l'intervalle [-1, 1]. Tandis que si le coefficient est proche de 0, les deux pixels ne sont pas reliés et ne peuvent pas être déduits l'un de l'autre.

Le Tableau IV.3 montre que l'algorithme du chaos possède des coefficients de corrélation verticale et diagonale presque équivalents à ceux de l'AES-CTR. Puisque notre chiffrement a été basé sur un découpage horizontal de l'image, nous nous concentrons sur le coefficient de corrélation horizontale qui est le plus représentatif. Les résultats calculés montrent que l'algorithme du chaos a un coefficient de corrélation horizontale plus faible que celui de l'AES-CTR. Ceci permet de conclure que l'algorithme du chaos est meilleur que l'AES-CTR dans la qualité de chiffrement d'une image.

		Coefficients de corrélation		
		horizontale	verticale	Diagonale
Image	initiale	0,9193	0,8724	0,8409
	chiffrée avec AES-CTR	0,0130	0,0250	0,0261
	chiffrée avec l'algorithme du chaos	-0,0037	0,0321	0,0262

Tableau IV.3 : Coefficients de corrélation des trois images.

IV.2.1.3.2.3 NPCR et UCAI

Pour le troisième temps de notre démarche de comparaison, nous avons calculé deux paramètres (NPCR et UCAI) largement utilisés dans la communauté de chiffrement d'images afin d'analyser les méthodes de chiffrement contre les attaques différentielles [119][118].

La valeur de NPCR *(Number of Pixels Change Rate)* permet d'évaluer le pourcentage de changement entre l'image en clair et l'image chiffrée par les deux algorithmes. La valeur de UCAI *(Unified Average Changing Intensity)* permet d'évaluer la moyenne de changement d'intensité de pixels entre les deux images chiffrées par les deux algorithmes.

Soient $C_1(i,j)$ et $C_2(i,j)$ les positions (i,j) d'un pixel de l'image chiffrée avec l'AES-CTR et celle chiffrée avec l'algorithme du chaos, respectivement. Soient M la taille horizontale de l'image et N sa taille veticale. Nous définissons la matrice D de dimension $M \times N$ avec $D(i,j) = 1$ si $C_1(i,j) = C_2(i,j)$ et $D(i,j) = 0$ si $C_1(i,j) \neq C_2(i,j)$. Les formules de NPCR et UCAI sont données dans l'Équation IV.1 et l'Équation IV.2 respectivement.

$$NPCR = \frac{\sum_{ij} D(i,j)}{M \times N} \times 100$$

Équation IV.1 : Le pourcentage de changement de pixels entre deux images.

$$UACI = \frac{1}{M \times N} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \frac{|C_1(i,j) - C_2(i,j)|}{255} \times 100$$

Équation IV.2 : La moyenne de changement d'intensité de pixels entre deux images.

Les valeurs optimales de ces deux paramètres d'après [117] sont : $NPCR_{opt} = 99{,}61\%$ et $UACI_{opt} = 33{,}46\%$. Le Tableau IV.4 montre que l'AES-CTR et l'algorithme du chaos sont proches des valeurs optimales. Les deux algorithmes ont des pourcentages de NPCR et UCAI similaires.

	NPCR	UACI
AES-CTR	99,6094	32,9589
L'algorithme du chaos	99,6765	32,9541

Tableau IV.4 : Valeurs des paramètres NPCR et UCAI pour l'AES-CTR et l'algorithme du chaos.

IV.2.1.3.3 Comparaison des deux algorithmes

Nous avons évalué et comparé les performances des deux algorithmes symétriques AES-CTR et l'algorithme du chaos de [47]. Les résultats ont montré une supériorité de l'algorithme du chaos en termes de temps d'exécution ainsi d'une légère supériorité dans la qualité de chiffrement d'une image de référence. Notre comparaison partielle nous a conduit à préférer l'algorithme du chaos pour sa rapidité mais il reste à prouver qu'il est aussi robuste que l'AES-CTR en termes de résistance contre les attaques. Cette étape reste parmi nos perspectives. En attendant et pour rester cohérents avec les différents standards sans fil, nous avons choisi d'intégrer le standard AES-CTR dans notre proposition.

IV.2.2 Algorithmes asymétriques

Nous avons utilisé dans notre proposition deux algorithmes asymétriques basés sur les courbes elliptiques (ECDH et ECIES) détaillés dans la partie II.2.2.2.6. Notre implémentation de ces deux algorithmes est basée sur la version 2.0 de TinyECC [102], une librairie configurable pour les opérations d'ECC dans les RCSF. Cette librairie offre plusieurs options d'optimisation [121] qui peuvent être activées ou désactivées. Nous avons choisi *secp160r1* comme paramètres de la courbe elliptique déjà implémentée dans la librairie TinyECC. En effet, ces paramètres sont recommandés par SECG *(Standards for Efficient Cryptography Group)* [122]. Puis nous avons activé toutes les optimisations de TinyECC pour atteindre un maximum de rapidité en termes d'exécution avec un coût acceptable d'occupation mémoire.

Nous avons dû adapter l'algorithme ECDH de la librairie afin de fonctionner avec la méthode (DHWI) (voir partie III.1.1.3.1). La taille de la clé DH partagée est égale à 160 bits.

Puisque nous utilisons cette clé dans l'algorithme AES-CTR, seuls les premiers 128 bits de la clé sont utilisés pour le chiffrement. L'algorithme ECIES de la librairie utilise une simple opération de XOR pour le chiffrement symétrique afin d'assurer la confidentialité des données. Afin d'améliorer la sécurité de cet algorithme, dans notre implémentation, nous lui avons intégré l'AES-CTR. Nous utilisons dans la suite, la notation ECAES pour désigner l'utilisation de l'algorithme ECIES avec l'intégration de l'AES-CTR. Avant de procéder à l'évaluation globale des coûts induits par notre architecture sécurisée, nous allons nous consacrer dans la partie suivante à l'impact de cette intégration mesuré en termes de temps d'exécution, de consommation d'énergie et d'occupation de mémoire.

IV.2.3 Résultats d'implémentation

Dans cette partie, nous présentons, d'une part, l'évaluation des opérations symétriques d'AES-CTR, et d'autre part, celles des opérations asymétriques suivantes : (i) la phase d'initialisation de ECC, ECDH et ECAES, (ii) la génération des clés privées/publiques d'ECC, (iii) le calcul de la clé commune partagée entre deux nœuds en utilisant ECDH, et (iv) l'opération de chiffrement/déchiffrement de l'ECAES. Ces opérations sont exécutées et évaluées sur les cartes TelosB en termes d'empreinte mémoire, de temps d'exécution et de consommation d'énergie. La taille de la clé symétrique de l'AES-CTR est fixée à 128 bits en utilisant 10 tours *(rounds)* de chiffrement et la taille des clés asymétriques ECC est fixée à 160 bits selon les recommandations des standards.

IV.2.3.1 Occupation de mémoire

Afin d'obtenir l'occupation de code dans les mémoires ROM et RAM pour les différents algorithmes, nous avons utilisé le script *cheksize.pl* qui existe dans la distribution de TinyOS. La Figure IV.5 montre la taille en octets, d'une part, de l'algorithme symétrique AES-CTR (3,96% de ROM et 5,98% de RAM), et d'autre part, la taille de l'algorithme asymétrique ECAES qui l'intègre dans son code (38,31% de ROM et 26,32% de RAM). L'algorithme ECDH dédié au calcul de la clé commune partagée entre deux nœuds consomme 23,85% de ROM et 19,49% de RAM. L'occupation mémoire d'ECC est de 15,97% de ROM et 9,66% de RAM. Nous utilisons la notation ECC pour désigner un algorithme contenant un ensemble de fonctions qui permettent le stockage des paramètres des courbes elliptiques et la génération des clés publiques/privées. Cette génération est basée sur la multiplication scalaire des valeurs codées sur 20 octets afin d'obtenir des coordonnées appartenant à la courbe

elliptique choisie. Afin d'accélérer ces multiplications, les auteurs de TinyECC ont intégrés plusieurs optimisations en avantageant le temps d'exécution au détriment de l'empreinte mémoire. Ceci explique la grande consommation de mémoire (ROM et RAM) des algorithmes asymétriques comparant à l'algorithme symétrique AES-CTR.

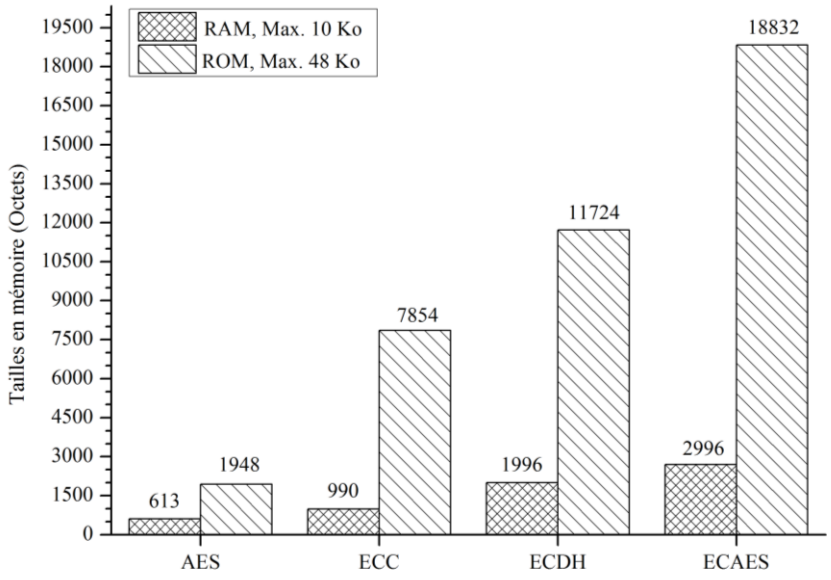


Figure IV.5 : Occupation en mémoire RAM et ROM des différents algorithmes.

IV.2.3.2 Temps d'exécution et consommation d'énergie

Nous avons mesuré le temps d'exécution et en avons déduit la consommation d'énergie des opérations de base pour l'AES-CTR ainsi que pour les trois algorithmes asymétriques ECC, ECDH et ECAES. Ces mesures de base sont : (i) la génération d'une clé privée/publique de 20 octets (ECC.gen), (ii) le calcul d'une clé commune selon DHWI de 16 octets (ECDH.cal), (iii) la phase d'initialisation d'AES-CTR (AES.init), et (iv) la phase d'initialisation de ECAES (ECAES.init). La Figure IV.6 montre le temps d'exécution ainsi que la consommation d'énergie de chacune de ces opérations. Chaque valeur représente une moyenne de 10 mesures sur les cartes TelosB. Nous avons estimé la consommation d'énergie en faisant l'hypothèse que la consommation d'énergie est une constante multipliée par le temps d'exécution. Cette hypothèse est fondée sur les formules et considérations suivantes : l'énergie consommée peut être estimée en utilisant la formule : U × I × t basée sur le temps d'exécution (t), la tension (U) et l'intensité du courant (I). Pour les cartes TelosB, nous avons fixé la tension à 3 Volts, en supposant que les piles sont toujours en charge complète (ou branchement sur un port USB, tension environ de 3 Volts). L'intensité du courant en mode

actif (avec composant radio désactivée) est de 1,8 mA [104]. L'intensité du courant lorsque le composant radio est en réception ou en émission est estimé à 23 mA.

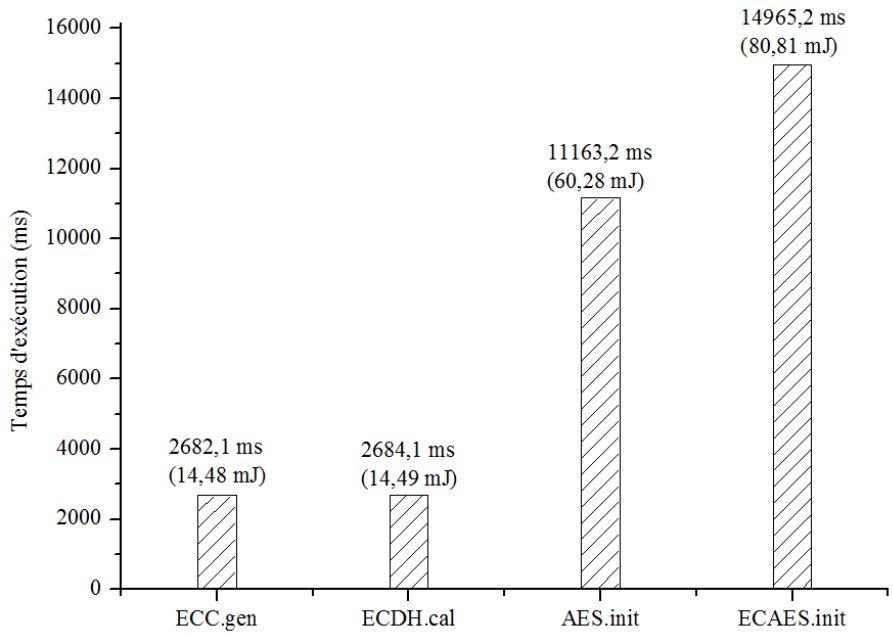


Figure IV.6 : Temps d'exécution de la phase d'initialisation des différents algorithmes. La consommation d'énergie de chaque opération est mise entre parenthèses en mJ.

La Figure IV.7 montre le chiffrement/déchiffrement d'un seul octet de données avec AES-CTR et ECAES. Nous remarquons que le chiffrement et le déchiffrement d'AES-CTR sont presque égaux. Cela revient à l'utilisation de l'AES en mode compteur. En effet, ceci est normal puisque nous utilisons les mêmes opérations pour chiffrer et déchiffrer. Tandis que le temps de chiffrement de l'ECAES est supérieur de 123,52 ms par rapport au temps de déchiffrement. Cela est dû à la création de clés temporaires afin d'assurer l'intégrité et la vérification du bloc envoyé du côté de l'expéditeur.

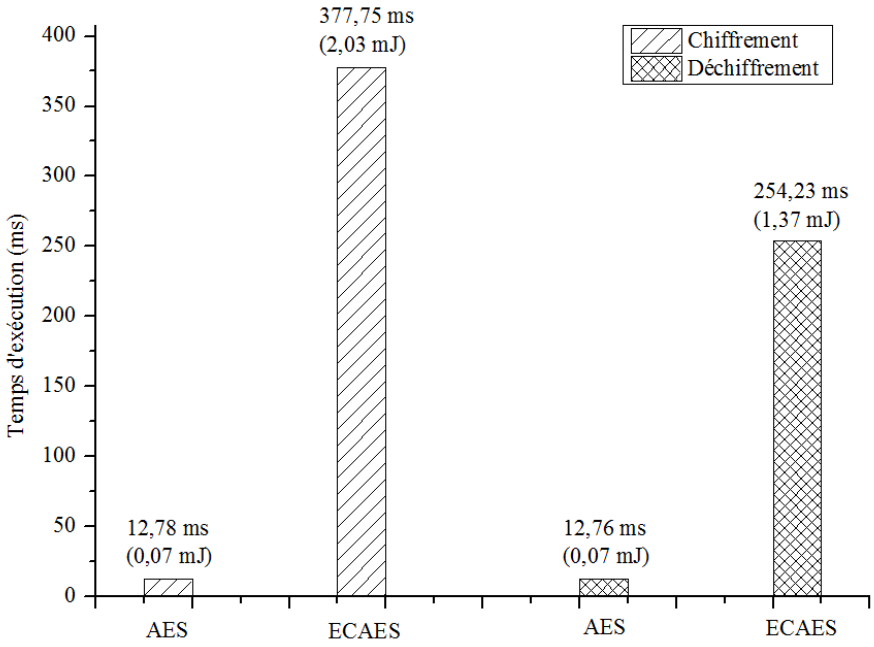


Figure IV.7 : Temps de chiffrement/déchiffrement d'un seul octet de données. La consommation d'énergie de chaque opération est mise entre parenthèses en mJ.

IV.3 Évaluation de notre proposition

Dans cette partie, nous évaluons le temps de calcul et la consommation d'énergie des trois phases principales de notre proposition : *JoinNet*, *NeighbDisc*, et la révocation et le renouvellement de clés. Nous commençons d'abord par évaluer l'occupation de la mémoire des nœuds avant et après le déploiement.

IV.3.1 Occupation de mémoire

Afin d'évaluer la taille du code des algorithmes que nous avons utilisés dans notre implémentation, nous additionnons uniquement la taille de l'ECDH et l'ECAES (Figure IV.5). En effet, l'ensemble des fonctions d'ECC sont déjà intégrées dans ECDH comme dans ECAES, et le code d'AES-CTR est intégré dans ECAES. Pour cette implémentation nous obtenons pour le code d'un nœud, l'empreinte mémoire suivante : 30 556 octets de ROM (62,16%) et 4 692 octets de RAM (45%). Notons que ces chiffres tiennent compte des 100 octets de clés pré-distribuées dans la mémoire des nœuds avant le déploiement (clé privée du nœud : 20 octets, clé publique du nœud : 40 octets et clé publique de la station de base : 40 octets). Cependant, nous avons constaté que le système d'exploitation TinyOS a besoin d'un minimum d'applications de démarrage de 1 396 octets de ROM et de 4 octets de RAM. Ainsi,

dans notre proposition un nœud utilisait, avant son déploiement, 31 952 octets de ROM (65%) et 4 696 octets de RAM (45,85%). Nous nous basons sur la Figure III.8 afin d'évaluer le nombre de clés stockées dans la mémoire des nœuds après leur déploiement. Soit *n* le nombre de voisins d'un nœud après son déploiement. Ainsi, le nombre maximum de clés qu'il doit stocker dans sa mémoire RAM est égale à $(2 * 16 + 56 * n)$ octets, où 16 octets désignent à la fois la taille d'une clé DH et celle de la clé commune du réseau et 56 octets désignent la taille d'une clé publique d'un nœud voisin (40 octets) additionnée à la taille de la clé symétrique partagée avec lui (16 octets). Ainsi à partir de cet exemple, 50,63% de la mémoire RAM seront occupés après le déploiement pour un nœud à 10 voisins quelle que soit la densité du réseau. La Figure IV.8 illustre le pourcentage approximatif de notre proposition en termes d'occupation de RAM et ROM des nœuds après le déploiement. Nous sommes conscients du fait que la taille d'une implémentation peut variée selon la façon de programmer et les techniques utilisées pour structurer le code ou optimiser son temps d'exécution par exemple, néanmoins ces chiffres nous donnent un ordre de grandeur significatif de l'empreinte mémoire d'une telle architecture sécurisée.

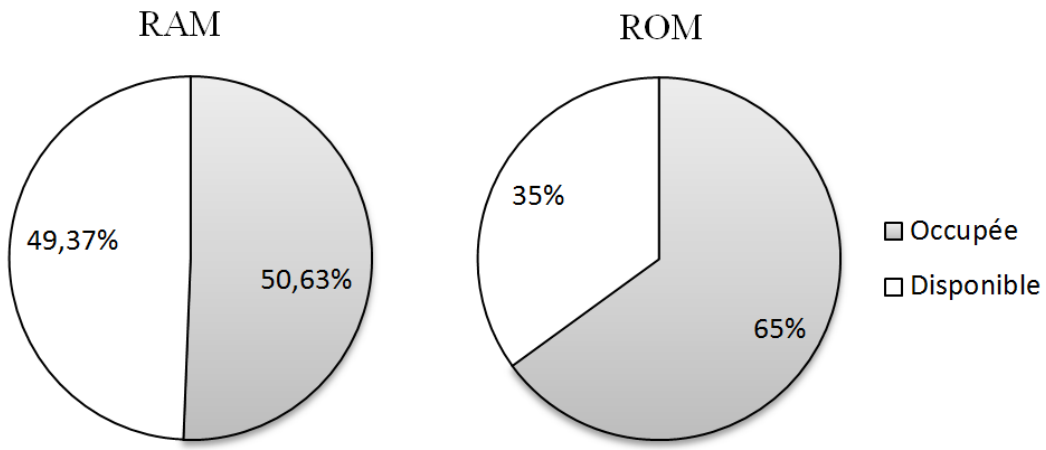


Figure IV.8 : Occupation de mémoire de notre proposition après le déploiement d'un nœud à 10 voisins.

IV.3.2 Temps d'exécution et consommation d'énergie

Dans cette partie, nous présentons l'évaluation en termes de temps d'exécution et de consommation d'énergie des phases : *JoinNet*, *NeighbDisc*, du renouvellement des clés symétriques et asymétriques et de la révocation de clés.

L'évaluation de notre proposition va être faite en considérant les deux types de topologies introduites dans la Figure IV.9 : l'une hiérarchique (a) et l'autre plate (b). D'après

la partie III.1.1.1 ces deux topologies peuvent être représentées sous forme de branches de chemins sécurisés exploitant chacune un lien direct et un ou plusieurs liens indirects par rapport à la station de base S. Quelle que soit la nature de la topologie, cette dernière sera considérée comme un ensemble des branches d'un arbre ayant S comme racine (c). Les nœuds N_i, N_j, N_k de la figure (c) sont des nœuds directs. Tandis que les sous-ensembles de nœuds $(N_{i+1}, N_{i+2}, \dots)$, $(N_{j+1}, N_{j+2}, \dots)$ et $(N_{k+1}, N_{k+2}, \dots)$ sont des nœuds indirects. Ils sont connecté à S via les nœuds directs N_i, N_j et N_k respectivement. Pour évaluer les différentes phases de notre proposition, nous avons besoin de considérer soit un échange d'un nœud avec un de ces voisins, soit un échange avec la station de base. L'évaluation faite sur une seule branche nous permettra d'obtenir des résultats transposables pour une évaluation complète des deux topologies. Dans la suite de ce chapitre, nous évaluons les phases de notre proposition sur une seule branche.

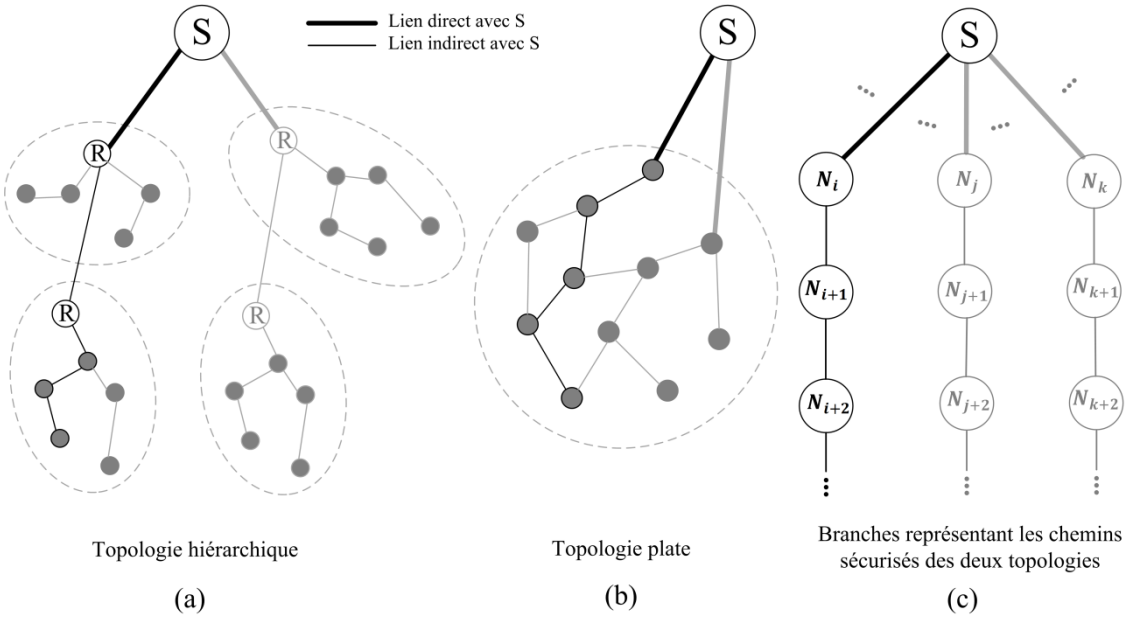


Figure IV.9 : Représentation des chemins des échanges nécessaires à notre solution pour les deux topologies de RCSF considérées.

Dans la Figure IV.10, nous présentons un exemple d'un RCSF constitué d'une branche de neuf cartes TelosB. Elles sont reliées via des concentrateurs USB à des ports d'un ordinateur portable afin de les alimenter et récupérer les résultats des évaluations. L'une des cartes est choisie pour jouer le rôle de la station de base. Les huit autres cartes constituent avec la station de base une branche de nœuds d'un RCSF.

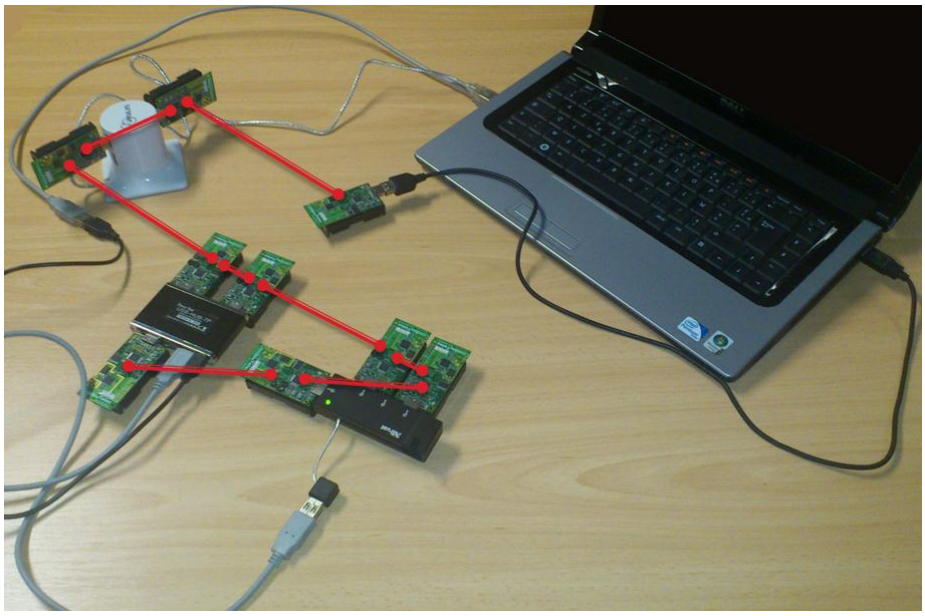


Figure IV.10 : Maquette d'évaluation d'un RCSF constitué d'une branche de neuf cartes TelosB.

IV.3.2.1 La phase *JoinNet*

Afin d'évaluer la phase *JoinNet*, nous avons mesuré le temps d'exécution de l'association au réseau d'un ensemble de nœuds. La Figure IV.11 montre l'association de 8 nœuds $\{N_i, 1 \leq i \leq 8\}$, le premier nœud N_1 rejoint directement le réseau via la station de base (N_0) tandis que les autres nœuds rejoignent le réseau indirectement. L'association du nœud N_i, $2 \leq i \leq 8$ est faite à partir du nœud N_{i-1}. Les chiffres de la figure représentent une moyenne de 20 mesures de la phase. Nous constatons que le nœud à portée se distingue des nœuds qui ne le sont pas ainsi qu'une forte variation du temps d'exécution de la phase *JoinNet* qui augmente en même temps que la taille du réseau. Le temps n'augmente pas d'une façon linéaire avec l'augmentation du nombre de nœuds. L'écart type calculé à partir des 20 mesures faites sur les 8 nœuds, varie entre 138,09 ms et 147,14 ms. Afin de découvrir la source de ces écarts-types importants, nous étudions dans la suite le temps nécessaire à l'éxécution de chacunes des phases élémentaires qui constituent la phase *JoinNet*.

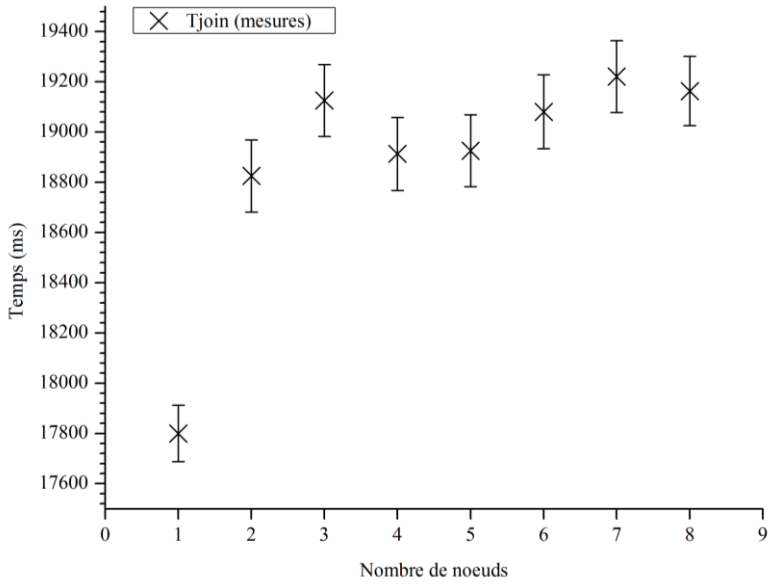


Figure IV.11 : Temps d'association des nœuds au réseau. Le nœud N_i est représenté par son indice *i* sur l'axe des abscisses.

Nous avons décomposé l'évaluation de la phase *JoinNet* en deux parties. La première concerne le calcul du temps d'exécution du processeur sur chaque nœud, et la deuxième le temps nécessaire pour envoyer et recevoir les paquets entre les nœuds. La Figure IV.12 montre la décomposition de la phase en tâches élémentaires de temps T_i où $1 \leq i \leq 5$. Dans notre exemple, nous supposons que le nœud R souhaite rejoindre le réseau par l'intermédiaire du nœud F, et que S est la station de base du réseau. Nous utilisons ces tâches élémentaires telles que :

- T_1 : temps dont R et S ont besoin pour calculer la clé secrète DH_{RS} partagée entre eux.
- T_2 : temps nécessaire pour chiffrer/déchiffrer un message contenant la requête *JoinReq* (la partie chiffrée de ce message est de 16 octets) en utilisant l'AES-CTR avec la clé commune DH_{FS} déjà établie entre S et F durant la phase *JoinNet* de F. *JoinReq* contient l'adresse physique et l'adresse logique de R ainsi qu'une valeur *nonces* n_R crée par R.
- T_3 : temps nécessaire pour chiffrer/déchiffrer un message contenant la réponse *JoinRes* qui contient la clé publique de R (la partie chiffrée de ce message est de 40 octets). Le chiffrement/déchiffrement de ce message est fait en utilisant l'AES-CTR avec la clé DH_{FS}. Notons que *JoinRes* contient la valeur n_R envoyé par R à S.
- T_4 : temps nécessaire pour chiffrer la clé symétrique SK_{FR} dédiée aux chiffrements de messages de données entre F et R avec ECAES.

- T_5 : indique le temps nécessaire pour déchiffrer SK_{FR} et *JoinRes* en utilisant ECAES.

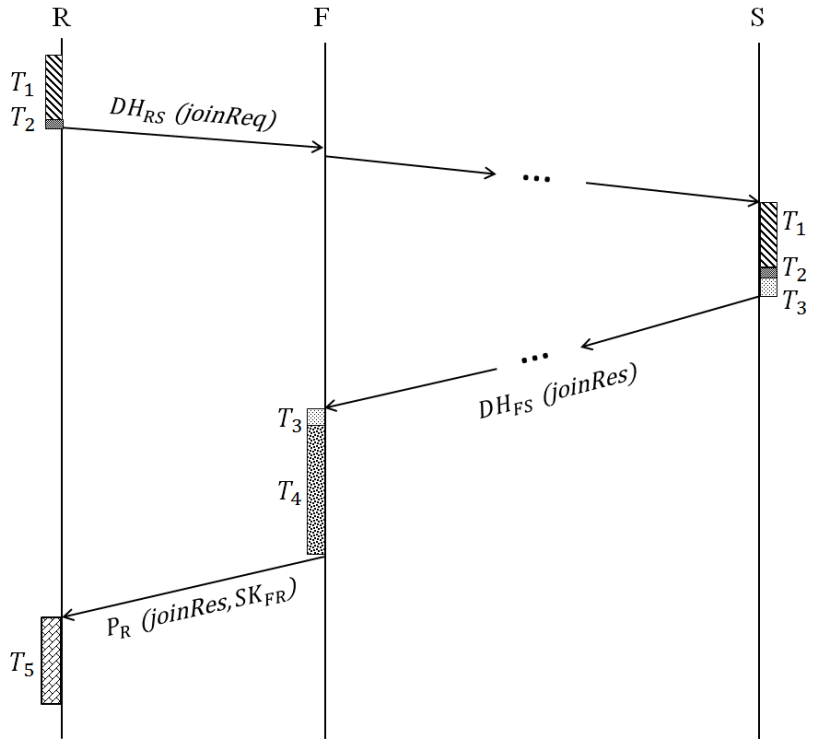


Figure IV.12 : Décomposition de la phase *JoinNet* en tâches élémentaires de temps d'exécution.

Dans notre démarche, il s'agit d'évaluer les contributions en termes de durée et d'écart type des différentes tâches élémentaires d'exécution de cette décomposition sur la durée totale de chaque phase. Ceci impose de faire de nouvelles mesures qui caractérisent le comportement de chaque période d'exécution en durée (moyenne et écart type). Les résultats obtenus sont consignés dans le Tableau IV.5. Les écarts types de ce tableau sont associés aux moyennes des durées des tâches élémentaires sur les 8 nœuds. Nous constatons que le chiffrement asymétrique est la principale cause des grands écarts types de la Figure IV.11. L'addition des écarts types des tâches élémentaires asymétriques est égale à 146,69 ms. Ce cumul constitue la plus grande partie de l'écart global, tandis que les tâches élémentaires de chiffrement symétrique couvrent le reste avec des écarts types négligeables par rapport aux tâches élémentaires de chiffrement asymétrique. Les tâches élémentaires T_4 et T_5 utilisent l'algorithme ECAES. Cet algorithme comprend la création d'une clé publique temporaire afin d'en dériver deux autres clés. La première est dédiée au chiffrement/déchiffrement (en utilisant AES) de la clé envoyée, et la deuxième est utilisée pour fabriquer le MAC (code) du message. Notons que la création d'une clé publique en utilisant ECC est faite en multipliant une clé privée (choisie aléatoirement dans un intervalle) par le point *G* de la courbe elliptique

utilisée. Ensuite, un test est fait pour vérifier si le point résultant appartient à la courbe (en fait à l'ensemble des points qui représentent la courbe). Si ce test est positif, le point obtenu est la clé publique recherchée. Sinon, une autre clé privée sera choisie jusqu'à ce que le résultat du test soit positif. Par conséquent, les écarts types de T_4 (141,87 ms) et T_5 (5,52 ms) s'expliquent par l'usage d'opérations utilisant une génération de clés, ils représentent l'essentiel de l'écart type global.

	T_1 (N_i)	T_2 (N_i)	T_5 (N_i)	T_1 (S)	T_2 (S)	T_3 (S)	T_3 (N_{i-1})	T_4 (N_{i-1})
Moyenne en ms	3 364,89	206,03	4320,96	3135,24	205,72	1345,87	612,91	6439,34
Ecart type	1,28	0,45	5,52	1,31	0,42	0,45	0,51	141,87

Tableau IV.5 : Ecart type des tâches élémentaires de la phase *JoinNet*.

Rappelons que l'algorithme ECDH utilise les multiplications scalaires pour obtenir la clé commune DH. La période T_1 dédiée à cette création peut différer d'un nœud à un autre puisqu'elle est obtenue des deux côtés en multipliant sa clé privée par la clé publique du partenaire. Ceci explique la variation de durée entre $T_1(N_i)$ (3 364,89 ms en moyenne) et $T_1(S)$ (3 135,24 ms en moyenne) dans le Tableau IV.5. Le Tableau IV.6 illustre cette variation en termes de moyennes et d'écarts types. Dans la suite de ce mémoire, nous faisons une moyenne des deux valeurs de T_1 afin d'élaborer les équations.

Nœud/S	1/S	2/S	3/S	4/S	5/S	6/S	7/S	8/S
Moyenne en ms	3 390,1/ 3 142,5	3 300,6/ 3 143,9	3 435,1/ 3 133,35	3 307,3/ 3 133,8	3 293,05/ 3 143,3	3 369,45/ 3 121,85	3 419,2/ 3 134,7	3 404,3/ 3 128,55
Ecart type	1,07/1,36	1,98/1,77	1,21/1,09	0,73/0,7	0,69/2,13	1,36/0,75	2,19/1,53	0,8/1,19

Tableau IV.6 : Moyennes et écarts types de 20 mesures de la période T_1.

Afin d'obtenir le coût total en termes de temps d'exécution et de consommation d'énergie de la phase *JoinNet* en fonction de la taille du réseau, nous présentons dans la suite l'élaboration d'une équation prenant en compte le cumul des étapes de la phase.

Nous présentons d'abord dans la Figure IV.13 le temps nécessaire à l'exécution et la consommation d'énergie correspondante des tâches élémentaires T_i. Chaque valeur de la figure représente une moyenne de 20 mesures de la même opération sur huit nœuds.

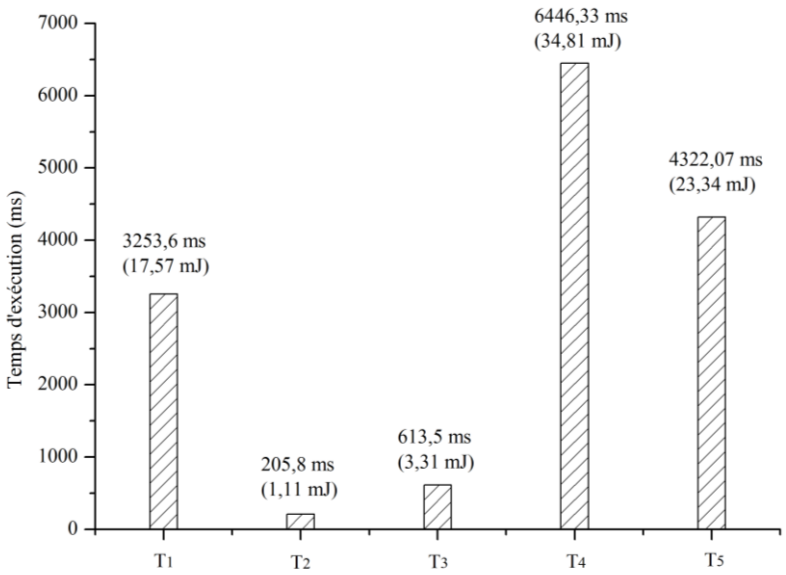


Figure IV.13 : Temps d'exécution des différentes tâches élémentaires de la phase *JoinNet*.

La taille maximale d'un paquet est de 133 octets (format IEEE 802.15.4), dont 19 octets sont déjà exploités par TinyOS. Il reste donc 114 octets utilisables pour notre implémentation. Nous avons dédié 5 octets pour un entête contenant des adresses logiques des nœuds sources et destinations (i est une adresse logique du nœud N_i et ce dernier est le $i^{ème}$ nœud qui a rejoint le réseau). Les 109 octets restants forment la taille maximale du champ de données d'un paquet. Le temps d'envoi et de réception des paquets doit être pris en compte. Les valeurs sont obtenues à partir des mesures réelles sur les cartes TelosB. Les entêtes sont transmis en clair accompagnées du champ de données qui peut être variable d'un envoi à un autre. Le temps nécessaire pour envoyer ou recevoir un paquet de 16 octets (*joinReq*) est de 12,53 ms, nommé S_{16}. L'envoi ou la réception d'un paquet de 40 octets (*joinRes* contenant la clé publique d'un nouveau nœud) prend 16,36 ms et sera nommé S_{40}. Le temps nécessaire pour envoyer ou recevoir 56 octets (*joinRes* contenant : l'abscisse d'une clé publique temporaire de 20 octets servant à créer un MAC lui-même de 20 octets et la taille d'une clé symétrique de 16 octets) est de 18,62 ms. Nous avons mesuré la durée entre l'instant où l'envoi du message est demandé et l'instant où le message est envoyé au médium. Notons que le temps d'exécution de la commande d'envoi prend plus de temps que la durée de la transmission. Dans le cas d'envoi ou de réception de 16, 40 et 56 octets, seulement 0,5 ms, 1,25 ms et 1,75 ms respectivement sont consommées pour transmettre le paquet sur le médium (ce calcul ne prend pas en compte le délai d'attente pour l'accès au médium mais prend en compte les entêtes de 19 et 5 octets qui viennent d'être définis).

Soit T_{join} le temps total obtenu par cumul des étapes intermédiaires de la phase de *JoinNet*, T_{join} pour le nœud F, donné par l'Équation IV.3. Dans cette équation, les T_i sont les durées moyennes de l'exécution des tâches élémentaires présentées en Figure IV.13, et *n* est le nombre de nœuds intermédiaires situés sur le chemin entre F et S. Nous avons un envoi de *n*+1 messages contenant 16 octets entre F et S. Si nous ajoutons le premier message de 16 octets entre R et F, nous aurons un temps égale à $(n+2) * S_{16}$ à calculer. Nous avons ensuite la réception de *n*+1 messages contenant 40 octets entre F et S, ce qui justifie la prise en compte d'un temps de $(n+1) * S_{40}$. Nous avons enfin un seul message de 56 octets entre F et R dont la durée S_{56}.complète l'équation suivante.

$$T_{join} = 2 * (T_1 + T_2 + T_3) + T_4 + T_5 + (n+2) * S_{16} + (n+1) * S_{40} + S_{56}$$

Équation IV.3 : Temps nécessaire pour accomplir un *JoinNet*.

La consommation totale d'énergie de la phase *JoinNet* est estimée en remplaçant les T_i $(1 \leq i \leq 5)$ de l'Équation IV.3 par l'énergie consommée durant chaque période, valeur donnée entre parenthèses dans la Figure IV.13. S_{16}, S_{40} et S_{56} sont remplacés aussi par la consommation d'énergie correspondante. Notons que l'Équation IV.3 correspond à l'association au réseau des nœuds indirects. Pour obtenir la durée nécessaire et la consommation d'énergie des nœuds directs, il suffit de remplacer $(n+2)$ par 1 et supprimer $(n+1) * S_{40}$.

D'après l'Équation IV.3, nous pouvons calculer le coût de l'association au réseau de n'importe quel nœud. Nous présentons dans la Figure IV.14 les valeurs obtenues de l'équation en remplaçant *n* par le nombre de nœuds intermédiaires entre S et F de la Figure IV.12. Ainsi, nous pouvons voir que la courbe de la durée globale de *JoinNet* coupe tous les segments qui représentent les points mesurés associés à leur écart type, de chaque association de nœuds. Cette équation validée par ce constat, permet d'extrapoler d'une façon linéaire en fonction de la taille du réseau quand la branche de nœuds impliquée dans une association dépasse 8 nœuds.

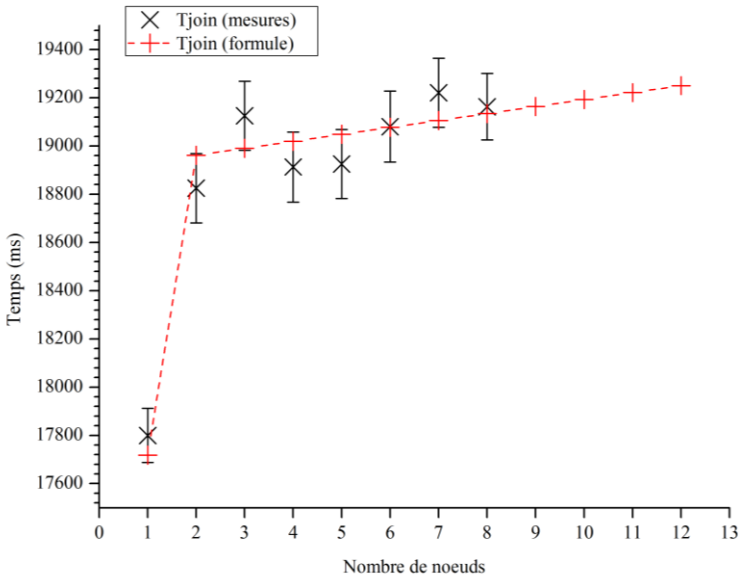


Figure IV.14 : Modélisation de la durée de la phase *JoinNet* en fonction de tâches élémentaires. Le nœud N_i est représenté par son indice *i* sur l'axe des abscisses.

IV.3.2.2 La phase *NeighbDisc*

La phase de découverte de voisinage peut être évaluée de la même manière que la phase *JoinNet*. La Figure IV.15 montre les mesures obtenues pour chaque nœud voulant créer une clé symétrique avec son voisin. Nous avons réalisé le test sur 8 noeuds $\{N_i, 1 \leq i \leq 8\}$ déjà associés au réseau par *JoinNET*. Alors N_i a déjà un lien sécurisé avec N_{i-1} et désire établir un autre lien sécurisé avec le nœud N_{i+1} avec la phase *NeighbDisc*. Les croix de la figure représentent une moyenne de 20 mesures. Nous avons calculé et ajouté sur la figure l'écart type des 20 mesures. Ces écarts types sont grands et varient entre 110,22 ms et 120,63 ms. Afin de découvrir la raison d'une telle garandeur et d'une telle variation, nous allons reconduire la méthode qui consiste à décomposer la durée de la phase de découverte de voisinage en durée d'exécution de phases élémentaires.

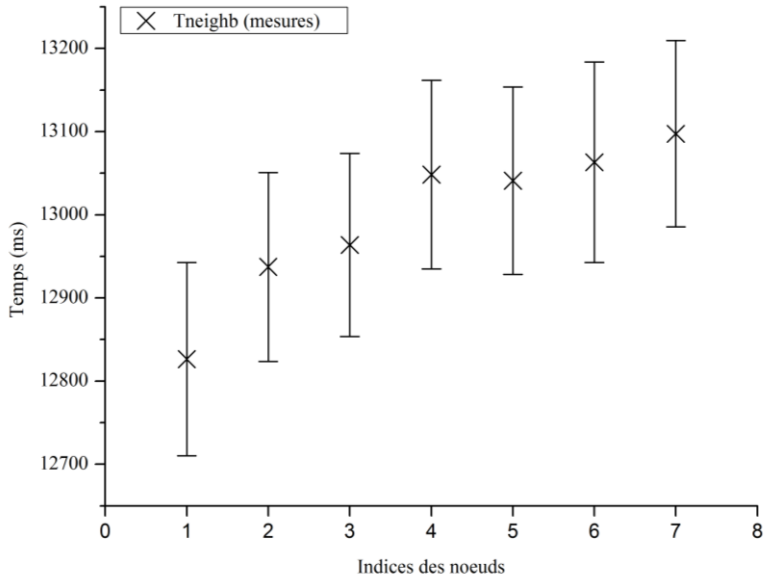


Figure IV.15 : Temps d'établissement d'un lien sécurisé d'un nœud N_i avec son nouveau voisin N_{i+1}.

Dans la Figure IV.16 le temps nécessaire à la phase *NeighbDisc* est décomposé en plusieurs durées élémentaires correspondant à l'exécution des différentes tâches élémentaires qui constituent cette phase de découverte. Ces tâches élémentaires sont les suivantes si l'initiative est prise par le nœud A :

- T_2 : temps nécessaire à A pour chiffrer/déchiffrer la requête *pkReq* (16 octets chiffrés) en utilisant AES-CTR avec la clé DH_{AS}. *pkReq* contient l'adresse physique et l'adresse logique de A ainsi qu'une valeur nonces n_A crée par A.
- T_3 : temps nécessaire pour chiffrer/déchiffrer un message contenant la réponse *pkRes* qui contient la clé publique de B (40 octets chiffrés) en utilisant AES-CTR avec DH_{AS}. *pkRes* contient aussi la valeur n_A envoyé par A à S.
- T_4 : temps nécessaire à A pour chiffrer la réponse *skEst* (contenant une clé de session dédiée aux chiffrements de messages de données entre A et B) avec ECAES. Ici *skEst* contient seulement l'adresse logique de A et une valeur *nonces* n'_A crée par A. Il pourra retrouver cette valeur dans l'ACK envoyé par B.
- T_5 : temps nécessaire pour déchiffrer SK_{FR} en utilisant ECAES.

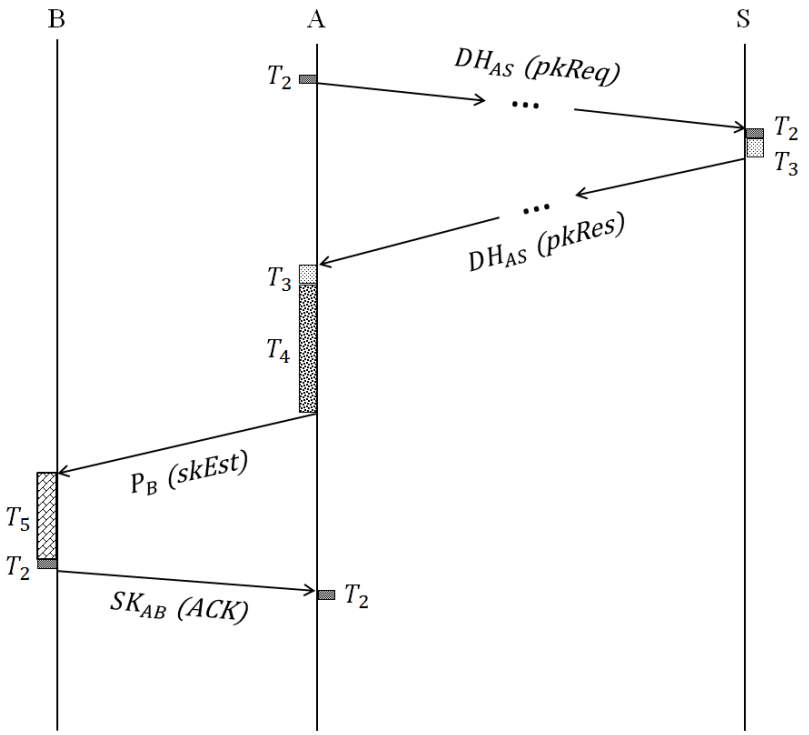


Figure IV.16 : Décomposition de la phase *NeighbDisc* en tâches élémentaires.

Les moyennes (et les écarts types associés) des temps nécessaires à l'exécution de ces tâches élémentaires sont présentées dans le Tableau IV.7. Ce tableau montre que l'écart type de la période T_4 représente presque la totalité de l'écart type global de la phase. Notons que T_5 possède un écart type de 4,58 ms et que les autres tâches élémentaires possèdent un écart type négligeable par rapport à T_4 et T_5. Nous pouvons faire la même constatation que celle que nous avions faite pour le Tableau IV.5 de *JoinNet*. Le chiffrement asymétrique d'ECAES utilise des opérations contenant une génération de clés publiques ECC qui consomment un temps à la fois long et variable.

	T_2 (N_i)	T_3 (N_i)	T_4 (N_i)	T_2 (S)	T_3 (S)	T_2 (N_{i-1})	T_5 (N_{i-1})
Moyenne en ms	206,1	614,06	6469,71	206,39	614,58	206,14	4326,96
Ecart type	0,23	0,53	112,73	0,45	0,44	0,29	4,58

Tableau IV.7 : Moyennes et écart type du temps d'exécution des tâches élémentaires de la phase *NeighbDisc.*

Soit T_{Neighb} le temps total nécessaire pour qu'un nœud puisse finir la phase de *NeighbDisc*. T_{Neighb} est donnée par l'Équation IV.4 où *n* est le nombre de nœuds intermédiaires situés sur le chemin entre A et S. Nous avons un envoi de *n*+1 messages contenant 16 octets entre A et S. Si nous ajoutons le dernier message de 16 octets entre B et

A, nous aurons un temps égal à $(n+2) * S_{16}$ à calculer. Il s'en suit une réception de n+1 messages contenant 40 octets entre A et S auxquels il faut faire correspondre un temps de $(n+1) * S_{40}$. Enfin, nous avons un seul message de 56 octets entre B et A, ce qui justifie l'ajout du temps de S_{56}.

$$T_{Neighb} = 4 * T_2 + 2 * T_3 + T_4 + T_5 + (n+2) * S_{16} + (n+1) * S_{40} + S_{56}$$

Équation IV.4 : Temps nécessaire pour accomplir *NeighbDisc*.

La consommation totale d'énergie de la phase *NeighbDisc* est calculée de la même manière que le temps d'exécution et cela en remplaçant les T_i $(2 \leq i \leq 5)$ de l'Équation IV.4 par l'énergie consommée pour chaque période présente entre parenthèses dans la Figure IV.13. S_{16}, S_{40} et S_{56} sont remplacés aussi par leur consommation d'énergie respective.

D'après l'Équation IV.4, nous pouvons calculer le coût de la découverte du voisinage *(NeighbDisc)* de n'importe quel nœud du réseau. Nous présentons dans la Figure IV.17 les valeurs obtenues de l'équation en remplaçant *n* par le nombre de nœuds intermédiaires entre A et S de la Figure IV.16. Ainsi, nous pouvons voir que la courbe coupe les segments représentant les valeurs moyennes et leur écart type de chaque phase découverte du voisinage, ce qui valide l'équation obtenue pour T_{Neighb}. Cette équation nous permet d'obtenir ce temps par extrapolation pour des réseaux de plus grande taille.

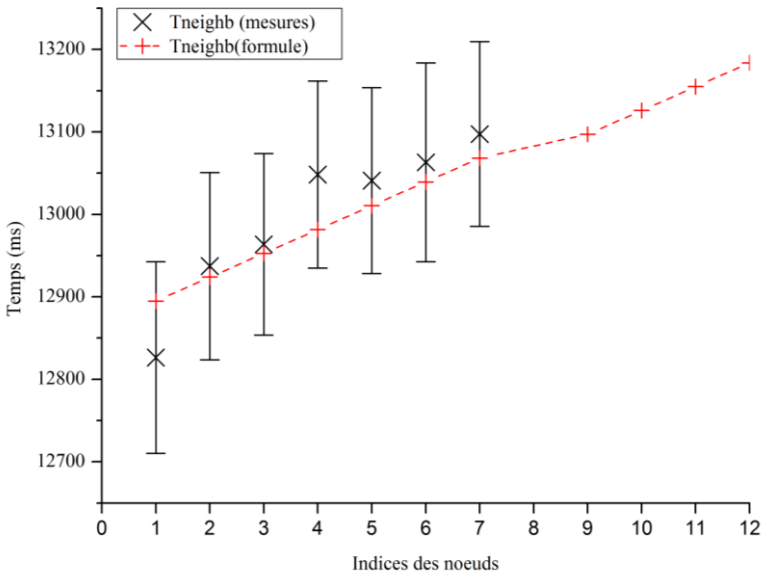


Figure IV.17 : Temps nécessaire à la découverte du voisinage.

IV.3.2.3 Le renouvellement de clés

Symétrique :

Nous avons mesuré le renouvellement d'une clé symétrique entre deux nœuds. Le coût total moyen donné par 20 mesures est de 11 135,35 ms avec un écart type de 113,35 ms. Afin de mieux comprendre ces chiffres nous allons comme pour les autres phases procéder à une décomposition en étapes élémentaires.

La Figure IV.18 montre la décomposition en tâches élémentaires pour le renouvellement d'une clé symétrique lancé par un nœud A. T_4 indique le temps nécessaire pour chiffrer le message de renouvellement *skRen* contenant une clé nouvelle (clé de session SK'_{AB}) dédiée aux chiffrements de messages de données entre A et B. Ce message est chiffré avec ECAES. T_5 indique le temps nécessaire pour déchiffrer SK'_{AB} en utilisant ECAES. Enfin, T_2 indique le temps nécessaire pour chiffrer/déchiffrer un acquittement de réception envoyé par B (16 octets chiffrés) en utilisant l'AES-CTR avec la clé SK'_{AB}. Après le déchiffrement réussi de l'acquittement de la part de A, les données futures seront chiffrées avec la nouvelle clé de session.

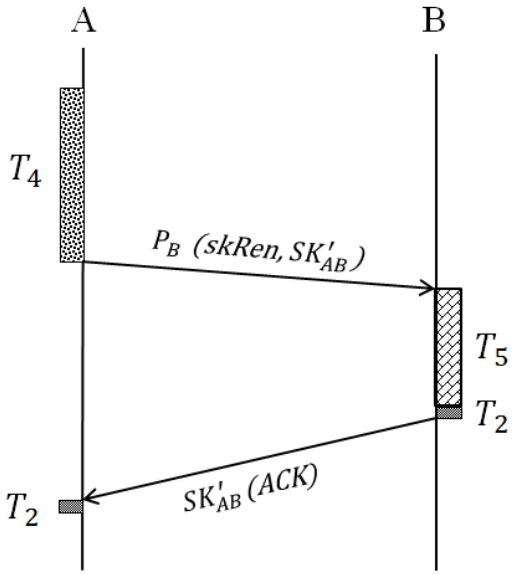


Figure IV.18 : Décomposition du renouvellement d'une clé symétrique.

Après la décomposition de la phase, nous avons mesuré le temps nécessaire à l'exécution de tâches élémentaires résultant de cette décomposition. Les temps d'exécution T_4 et T_5 sont à la base de l'écart type de cette phase. Ils ont un écart type de 112,99 ms et 4,02.ms

respectivement. L'écart type important de ces deux tâches, comme expliqué dans la phase *JoinNet*, est dû à l'utilisation des opérations pour une génération de clés publiques.

Soit T_{renSym} le temps total nécessaire pour qu'un nœud puisse réaliser un renouvellement de clé symétrique avec un voisin. T_{renSym} est donnée par l'Équation IV.5. Cette équation ne contient pas de variable, son coût est égal à 11,18 secondes, ce qui est un résultat très proche des mesures réalisées (11,13 secondes).

$$T_{renSym} = 2 * T_2 + T_4 + T_5 + S_{16} + S_{56}$$

Équation IV.5 : Temps nécessaire pour réaliser le renouvellement d'une clé symétrique.

La consommation totale d'énergie du renouvellement d'une clé symétrique est calculée de la même manière que le temps d'exécution et cela en remplaçant les T_i $(i = 2, 4, 5)$ de l'Équation IV.5 par l'énergie consommée de chaque tâche élémentaire donnée entre parenthèses dans la Figure IV.13. S_{16} et S_{56} sont remplacés aussi par leur consommation d'énergie respective.

Asymétrique :

Nous avons mesuré le renouvellement d'une nouvelle paire de clés asymétrique lancé par S à destination de 8 nœuds $\{N_i, 1 \leq i \leq 8\}$. La Figure IV.19 montre les valeurs moyennes issues de 20 mesures, représentées par des croix, dès l'envoi d'une nouvelle paire de clés jusqu'à la réception d'un acquittement du destinataire. Les écarts types de ces mesures varient entre 33,21 ms et 57,19 ms, nous allons les étudier en reprenant la méthode pratiquée précédemment.

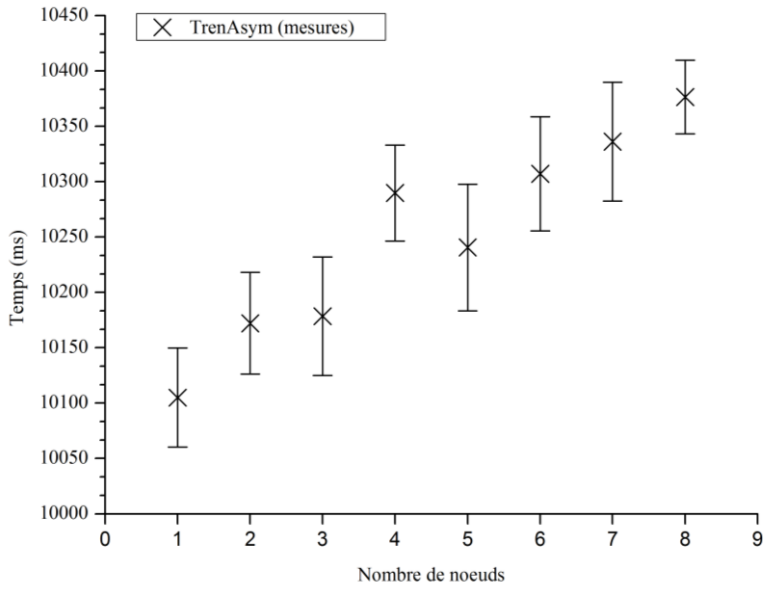


Figure IV.19 : Temps nécessaire au renouvellement d'une paire de clés asymétriques accompagnée de la clé publique de S. Le nœud N_i est représenté par son indice *i* sur l'axe des abscisses.

La Figure IV.20 montre la décomposition en tâches élémentaires du renouvellement d'une paire de clés asymétriques lancé par la station de base. T_6 indique le temps nécessaire pour chiffrer le message contenant la nouvelle paire de clés pour le nœud V (Figure IV.20) accompagné de la nouvelle clé publique de la station de base. Cette dernière permettra à V de calculer la nouvelle clé DH liée dans sa création à la nouvelle clé publique de S. Ce message est chiffré avec AES-CTR et il est de 100 octets. Le coût d'envoi ou de réception de ce message, nommé S_{100}, est de 28,31 ms. T_6 coûte 1 249,25 ms (moyenne de 20 mesures) et 7,72 mJ en temps d'exécution et en consommation d'énergie, respectivement. T_1 indique le temps dont S et V ont besoin pour calculer la nouvelle clé secrète DH_{SV} partagée entre eux. Enfin, T_2 indique le temps nécessaire pour chiffrer/déchiffrer un acquittement de réception envoyé par V (16 octets chiffrés) en utilisant AES-CTR avec la clé DH'_{SV}.

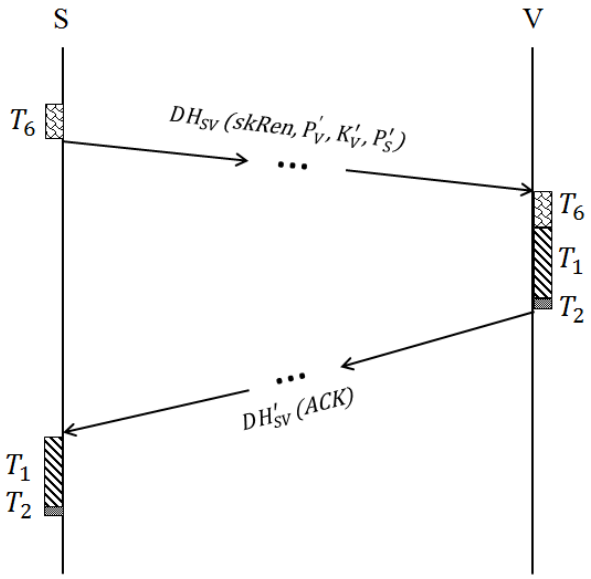


Figure IV.20 : Décomposition du renouvellement d'une paire de clés.

Le Tableau IV.8 montre les écarts types des tâches composant la phase de renouvellement de clés asymétriques. Nous constatons que la période T_1 (V) et T_1 (S) ont un écart type de 44,27 ms et 5,35 ms respectivement (voir phase *JoinNet* pour l'explication du fort écart type). Ceci justifie les valeurs des écarts types de la Figure IV.19.

	T_1 (S)	T_2 (S)	T_6 (S)	T_1 (S)	T_2 (S)	T_6 (S)
Moyenne en ms	3433,34	206,23	1429,24	3363,78	206,43	1429,33
Ecart type	5,35	0,42	1	44,27	0,49	0,7

Tableau IV.8 : Moyennes et écarts types des tâches de la phase de renouvellement de clés asymétriques.

Soit $T_{renAsym}$ le temps total nécessaire pour que la station de base puisse envoyer une nouvelle paire de clés asymétriques à un nœud V quelconque du réseau. $T_{renAsym}$ est donnée par l'Équation IV.6 où *n* est le nombre de nœuds intermédiaires situés sur le chemin entre S et V. Nous avons un envoi et une réception de *n*+1 messages contenant 100 octets et 16 octets respectivement entre S et V, ce qui nous conduit à la formulation suivante :

$$T_{renAsym} = 2 * (T_1 + T_2 + T_6) + (n + 1)(S_{16} + S_{100})$$

Équation IV.6 : Temps nécessaire pour réaliser le renouvellement d'une paire de clés asymétriques.

La consommation totale d'énergie du renouvellement d'une paire de clés asymétriques est calculée selon la méthode explicitée précédemment.

D'après l'Équation IV.6, nous pouvons calculer le coût de l'envoi d'une nouvelle paire de clés asymétriques accompagnée par la nouvelle clé publique de S à n'importe quel nœud du réseau. Les valeurs présentées dans la Figure IV.21 ont été obtenues de l'équation en remplaçant *n* par le nombre de nœuds intermédiaires entre S et V de la Figure IV.20. La courbe déduite de cette équation représente assez fidèlement les résultats obtenus par la mesure car elle coupe les segments représentant l'écart type de la plupart des points de mesure. Ainsi, nous pouvons par extrapolation estimer le comportement d'un réseau de plus grande taille.

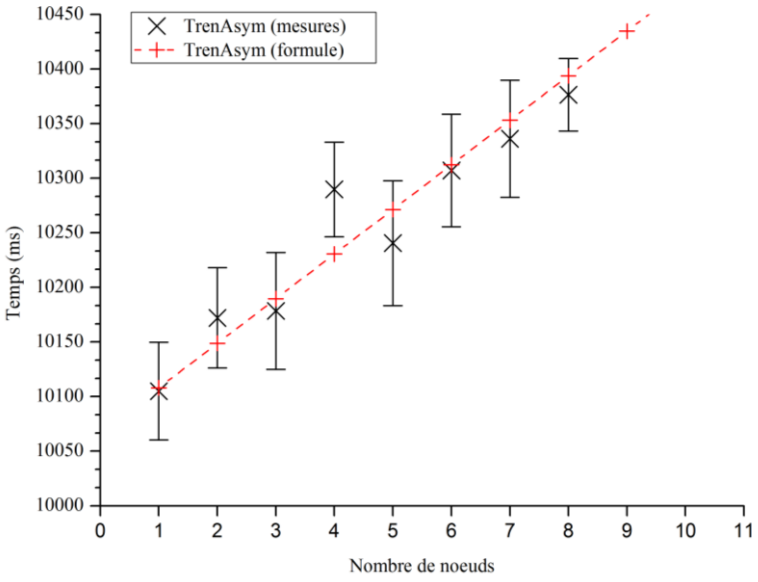


Figure IV.21 : Temps nécessaire au renouvellement asymétrique. Le nœud N_i est représenté par son indice *i* sur l'axe des abscisses.

Notons que le renouvellement de clés symétriques et asymétriques que nous avons proposé est destiné aux deux topologies : plate et hiérarchique. Le mécanisme de renouvellement de clés asymétriques est fait de la même manière dans les deux topologies. Il est lancé par la station de base. Le coût en temps d'exécution de ce mécanisme est évalué en calculant $T_{renAsym}$ de l'Équation IV.6. Le mécanisme de renouvellement de clés symétriques est réalisé différemment. Le coût d'un renouvellement d'une clé symétrique entre deux nœuds d'un réseau plat est évalué en calculant T_{renSym} de l'Équation IV.5, tandis que pour une topologie hiérarchique comme la topologie de MaCARI par exemple (voir Figure III.12), chaque routeur ou chef de cluster A est responsable du lancement du mécanisme de renouvellement symétrique. Ainsi, il crée une nouvelle clé de session pour son cluster avant de l'envoyer à B, un nœud de son cluster, comme le montre la Figure IV.18. Le coût total de

ce mécanisme, évalué par cluster, est $k * T_{renSym}$, où k est le nombre de nœuds du cluster. Le renouvellement de clés symétriques entre deux routeurs est réalisé de la même manière que pour deux nœuds d'un réseau plat avec un coût en temps égal à T_{renSym}.

IV.3.2.4 La révocation

Nous avons mesuré le temps nécessaire pour l'envoi d'un message de révocation de S à destination de 8 nœuds $\{N_i, 1 \leq i \leq 8\}$. La Figure IV.22 montre les valeurs moyennes issues de 20 mesures représentées par des croix. Les écarts types de ces mesures varient de 4,7 ms à 11,30 ms. Nous décomposons cette phase en tâches élémentaires pour retrouver l'origine des variations du temps nécessaire à cette action.

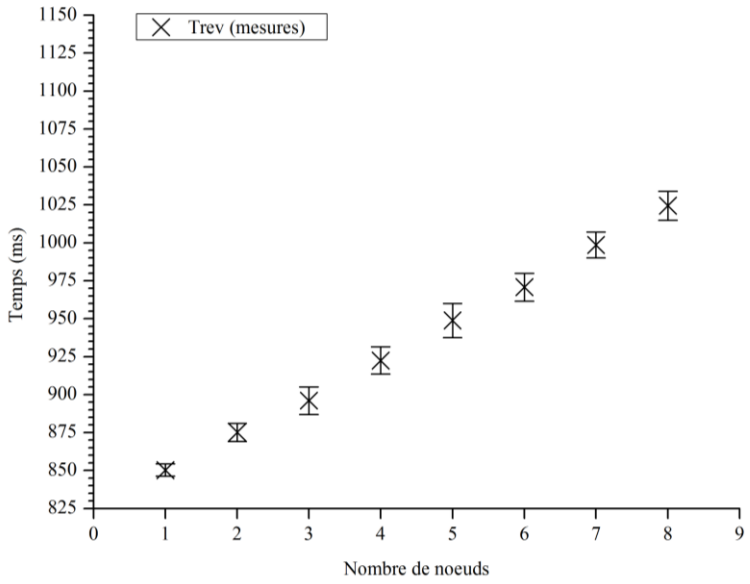


Figure IV.22 : Temps consommé pour l'envoi d'un message de révocation de S à 8 nœuds du réseau. Le nœud N_i est représenté par son indice i sur l'axe des abscisses.

La Figure IV.23 montre la décomposition en tâches de l'envoi d'un message de révocation lancé par la station de base aux voisins ayant un lien sécurisé avec un nœud capturé. Le message contient une demande de suppression (*revReq*) de liens avec le nœud capturé (voir Figure III.9). *revReq* contient aussi une valeur *nonces* n_S. T_2 indique le temps nécessaire pour chiffrer/déchiffrer le message de révocation ou l'acquittement de réception envoyé à ou par V (16 octets chiffrés) en utilisant AES-CTR avec la clé DH_{SV}. Après le déchiffrement réussi de l'acquittement de la part de S, elle vérifie tout d'abord que n_S extaite du déchiffrement est identique à celle qui est envoyée. Puis, elle lance le processus de création

d'une nouvelle paire de clés asymétriques (voir Figure IV.20). Notons que dans le cas de maintenance planifiée d'un nœud par exemple, S se peut se contenter d'envoyer uniquement le message de révocation sans lancer un processus de renouvellement de clés (pour cela, il faut qu'une telle action de maintenance soit signifiée à la station de base).

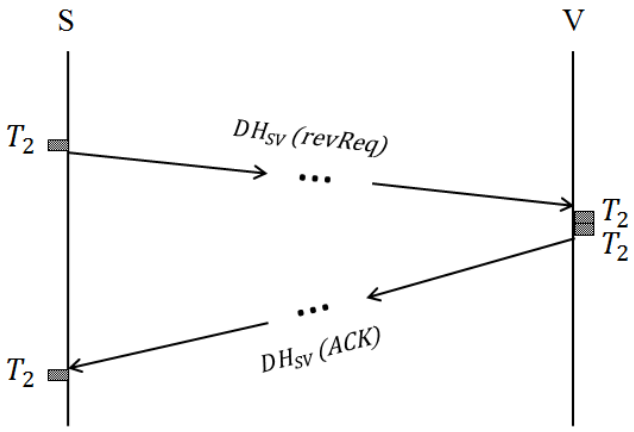


Figure IV.23 : Décomposition de l'envoi d'un message de révocation en cas de capture d'un nœud. S est la station de base et V est un nœud voisin du nœud capturé ayant un lien sécurisé avec lui.

Le Tableau IV.9 montre les valeurs moyennes de nos mesures ainsi que les écarts types des tâches composant l'envoi d'un message de révocation à une destination V. T2-rev (S), T2-ack (S), T2-rev (V) et T2-rev (V), représentent le temps d'exécution de tâches introduites dans la Figure IV.23. Ces temps ont des écarts types faibles, égaux à 0,21, 0,37, 0,47 et 0,47 ms respectivement. Nous remarquons que les écarts types de ces quatre tâches ne peuvent expliquer l'ampleur des écarts types globaux de l'envoi. Nous pouvons conclure des résultats là-aussi que le chiffrement symétrique prend un temps quasi-constant puisqu'il est associé à un faible écart type. Nous retrouvons l'explication de l'écart type global dans la formule de l'Équation IV.7. C'est la variation du temps nécessaire à la transmission qui a un impact majeur sur l'écart type global.

	T_2-rev (S)	T_2-ack (S)	T_2-rev (S)	T_2-ack (S)
Moyenne en ms	205,95	205,86	205,97	206,03
Ecart type	0,21	0,37	0,47	0,47

Tableau IV.9 : Valeurs moyennes et écarts types du temps nécessaire à l'envoi d'un message de révocation de S vers V, où V représente le nœud destination.

Soit T_{rev} le temps total nécessaire pour que la station de base puisse envoyer et recevoir la confirmation de réception d'un message de révocation à un nœud V. T_{rev} est donnée par l'Équation IV.8 où *n* est le nombre de nœuds intermédiaires situés sur le chemin entre S et V. Nous avons un envoi et une réception de *n*+1 messages contenant 16 octets entre S et V. Ainsi, nous aurons un temps égal à $2 * (n + 1) * S_{16}$ à prendre en compte.

$$T_{rev} = 4 * T_2 + 2 * (n + 1) * S_{16}$$

Équation IV.8 : Temps nécessaire pour envoyer et acquitter un message de révocation.

La consommation totale d'énergie de l'Équation IV.8 est calculée de la même manière que précédemment en remplaçant ici T_2 par 1,1 mJ.

Soit *k* le nombre de voisins d'un nœud capturé ou parti, le coût total du mécanisme de révocation en termes de consommation de temps et d'énergie est $k * T_{rev}$.

Notons que le renouvellement de la clé commune du réseau a un coût égal à $n * T_{rev}$, où *n* est le nombre de nœuds du réseau. En effet, la nouvelle clé commune est envoyée par S en *unicast* chiffré par la clé DH du destinataire. Et le chiffrement d'un message de révocation donne un résultat sur 16 octets (taille d'une nouvelle clé symétrique) et chiffré par la clé DH du destinataire.

D'après l'Équation IV.8, nous pouvons évaluer le coût de l'envoi d'un message de révocation envoyé par S à n'importe quel nœud du réseau. La Figure IV.24 présente les valeurs obtenues de l'équation en remplaçant *n* par le nombre de nœuds intermédiaires entre S et V de la Figure IV.23. La courbe obtenue à partir de l'équation représente fidèlement les résultats obtenus par des mesures et nous permet d'estimer ce temps lié à la révocation par extrapolation pour des réseaux de plus grande taille.

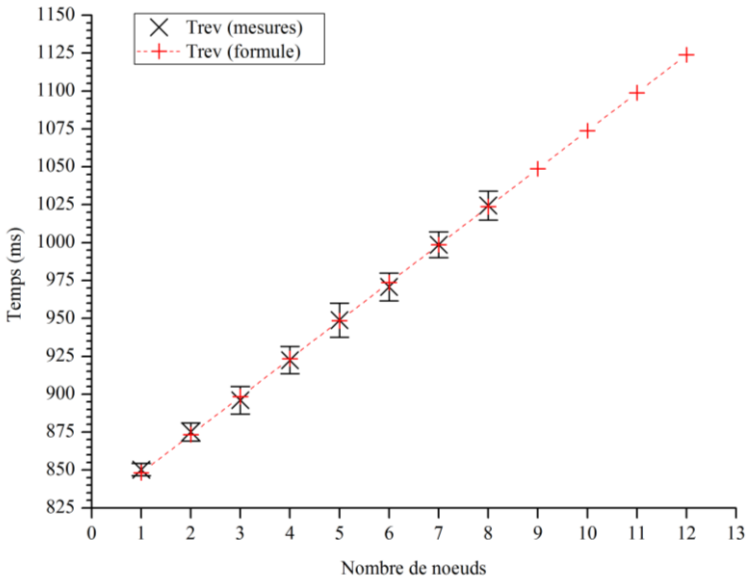


Figure IV.24 : Temps nécessaire à l'envoi d'un message de révocation. Le nœud N_i est représenté par son indice i sur l'axe des abscisses.

IV.3.2.5 Le chiffrement/déchiffrement des données

Nous avons mesuré le temps de chiffrement avec une clé symétrique de bout-en-bout des données envoyées à S par un nœud à un nombre de sauts variable. Les mesures de la Figure IV.25 montrent un exemple d'envoi de données de 16, 32, 48, 64 et 96 octets d'un nœud à un, deux, …, et huit sauts de S. À la réception, S envoi au nœud source un acquittement de 16 octets, chiffré avec la clé DH. La figure montre que les écarts types sont très faibles et les courbes tracées sur la figure se comportent de façon linéaire en fonction de l'augmentation du nombre de sauts.

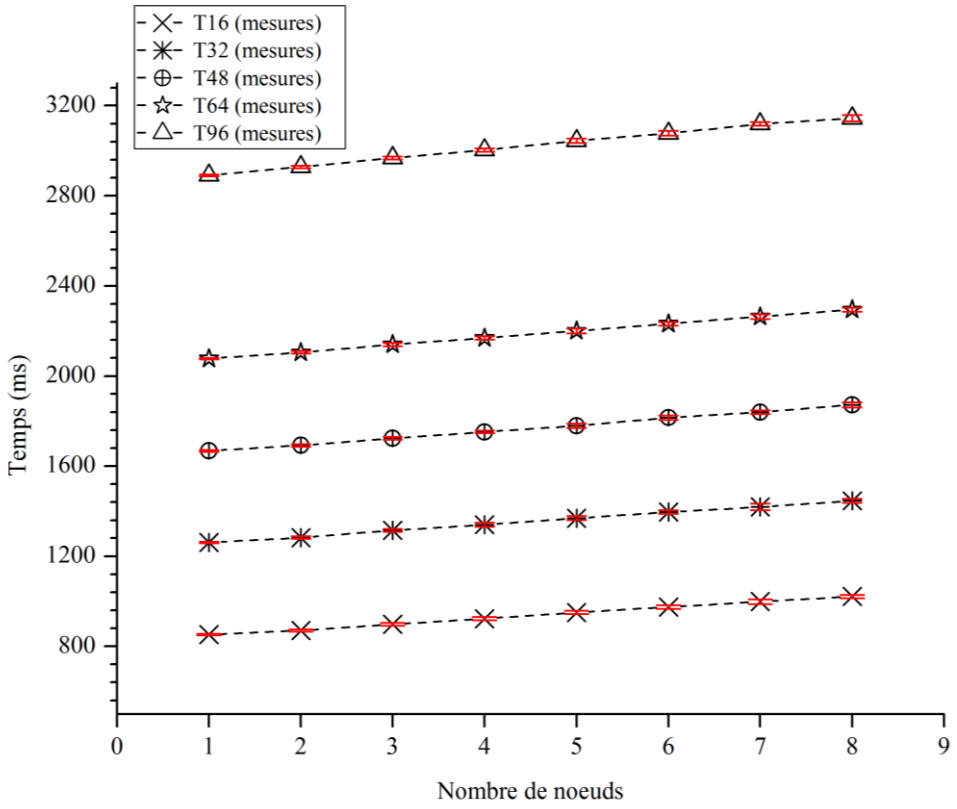


Figure IV.25 : Délai lié au chiffrement de bout-en-bout de 16, 32, 48, 64 et 96 octets de données. Le nœud N_i est représenté par son indice *i* sur l'axe des abscisses.

Notons que si les messages envoyés ont besoin d'être déchiffrés et chiffrés de nouveau par les nœuds intermédiaires entre la source de données et S, il suffit de calculer le coût selon la formule suivante : $T_{chiff\_unicast} = (n + 1) * T_i(1)$ où i = 16, 32, 48, 64 et 96 octets, n est le nombre de nœuds intermédiaires entre le nœud source et *S*, et $T_i(1)$ est le temps d'envoi des données de taille *i* d'un nœud à portée de *S*.

La Figure IV.26 montre l'IHM (Interface Homme Machine) de notre implémentation lorsqu'un nœud source (identité égale à 1) envoie 64 octets de données chiffrées à S (identité égale à 0). La figure montre le message avant et après le déchiffrement ainsi que le contenu du paquet envoyé et reçu. Le paquet est constitué d'un entête de 5 octets et d'un champ pour les données.

Nous avons fait le choix de ne pas chiffrer les entêtes dans notre solution pour ne pas alourdir d'avantage les temps de calcul. Dans cette implémentation, les nœuds intermédiaires entre une source et une destination peuvent router un paquet s'il ne leur est pas destiné, sans avoir le déchiffrer.

S'il est nécessaire de chiffrer les entêtes pour les protéger contre les attaques visant le routage, les entêtes peuvent être chiffrés avec la clé commune du réseau. Cette dernière est renouvelée régulièrement par *S*. Ainsi, les nœuds intermédiaires ont besoin seulement de déchiffrer l'entête, vérifier si le paquet leur est destiné et chiffrer de nouveau l'entête pour le prochain saut avec la clé commune du réseau avant de l'envoyer à ses voisins.

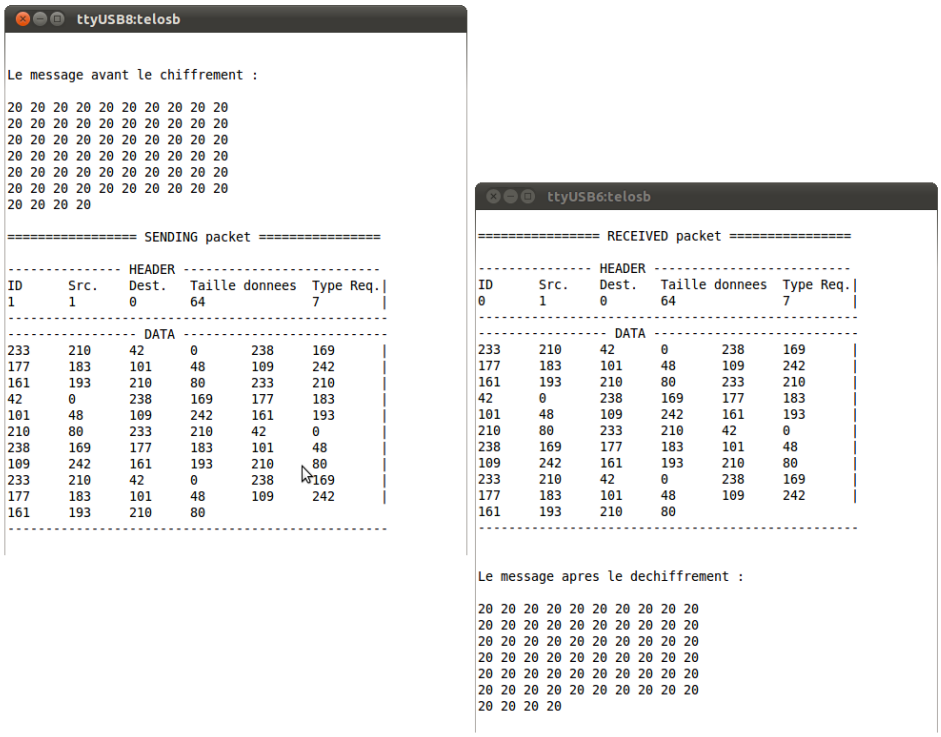


Figure IV.26 : Capture d'écran d'un chiffrement d'un paquet de données de 64 octets.

IV.3.3 L'effet de la taille du réseau

Dans cette partie, nous étudions l'effet de la taille du réseau en termes de nombre de sauts sur la consommation de temps et d'énergie. Lorsque le nombre de nœuds intermédiaires séparant les nouveaux arrivants de la station de base est grand, la consommation d'énergie et le temps d'exécution deviennent plus conséquents comme l'indique l'Équation IV.3. La même constatation s'applique aussi pour la phase de découverte des voisins, le renouvellement de clés asymétriques et l'envoi d'un message de révocation par la station de base comme l'indique l'Équation IV.4, l'Équation IV.6 et l'Équation IV.8 respectivement.

Dans notre proposition, le nombre d'opérations cryptographiques reste inchangé par rapport à l'augmentation du nombre de nœuds intermédiaires *n* entre un nœud et la station de base. En effet, nous utilisons le chiffrement de bout-en-bout. Les nœuds intermédiaires

transmettent les paquets sans avoir besoin de les déchiffrer ni de les chiffrer de nouveau. Ainsi, lorsque la taille du réseau augmente, seules les tâches élémentaires S_i ($i = 16,\ 40,\ 56,\ 100$) sont multipliées par des coefficients calculés en fonction de *n* comme le montrent les équations (voir Équation IV.3, Équation IV.4, Équation IV.6 et Équation IV.8).

Notons que durant les différentes phases de notre proposition, les nœuds ne sont pas nécésairement éveillés (la partie radio des nœuds) tout au long de l'exécution de ces phases. Ainsi, la consommation d'énergie varie selon le protocole MAC (d'accès au médium) et les cycles de travail de chaque nœud. Le temps mesuré de chaque phase représente la consommation de délais minimaux car nous n'avons pas pris en compte la perte de paquets, les répétitions ni le délai d'accès au médium. Ce délai est dépendant de la charge du réseau.

En l'absence de nœuds intermiédiaires, la phase de renouvellement d'une clé symétrique n'est pas affectée par la taille du réseau. Nous retrouvons son coût dans l'Équation IV.5.

IV.3.4 Comparaisons

Nous avons comparé dans le Tableau II.5 les schémas proposés pour sécuriser les RCSF selon plusieurs critères. Nous avons inclus notre proposition dans le Tableau III.1 afin de la comparer avec des schémas représentatifs en termes de passage à l'échelle, connectivité et résistance contre les attaques. Nous présentons dans le Tableau IV.10 la suite de cette démarche de comparaison avec des schémas représentatifs en termes d'utilisation de ressources. Les ronds du tableau désignent un défaut. Nous avons mis trois, deux ou un seul rond noir dans la colonne « Coût de ressources utilisées » pour noter que les schémas coûtent respectivement une forte, moyenne ou faible utilisation de ressources.

<table>
<tr><th colspan="2" rowspan="2">Schémas</th><th colspan="3">Critères de comparaison</th></tr>
<tr><th rowspan="2">Type de Schémas</th><th rowspan="2">Auteurs des schémas</th><th colspan="3">Utilisation de ressources</th></tr>
<tr><th>Mémoire (stockage de clés)</th><th>Temps d'exécution et consommation d'énergie</th><th>Renouvellement et révocation</th></tr>
<tr><td rowspan="4">Schémas à clé symétriques</td><td>Chan et al. [73]</td><td rowspan="2">●●●</td><td>●○○</td><td rowspan="3">●●●</td></tr>
<tr><td>Chan et Perrig (PIKE) [75]</td><td>●●○</td></tr>
<tr><td>Perrig et al.(SPINS) [80]</td><td rowspan="2">●○○</td><td rowspan="2">●○○</td></tr>
<tr><td>Zhu et al.(LEAP) [84]</td></tr>
<tr><td colspan="2">Notre proposition</td><td>●○○</td><td>●●○</td><td>●●○</td></tr>
<tr><td rowspan="4">Schémas à clé publiques</td><td>ZigBee (PKKE) [90]</td><td rowspan="4">●○○</td><td>●●●</td><td>●●○</td></tr>
<tr><td>Oliveira et al. (TinyPBC) [95]</td><td>●●○</td><td>●○○</td></tr>
<tr><td>Munivel et al. (micro-PKI) [88]</td><td rowspan="2">●●●</td><td>●●○</td></tr>
<tr><td>Watro et al. (TinyPK) [89]</td><td>●●●</td></tr>
</table>

Tableau IV.10 : Comparaison de notre proposition avec les méthodes existantes en termes d'utilisation de ressources. Dans ce tableau, le rond noir est utilisé pour évaluer un défaut.

Le tableau montre que notre proposition a un faible coût de stockage de clés comme les schémas asymétriques et quelques schémas symétriques, tandis qu'elle a un coût équivalent à TinyPBC en temps d'exécution et consommation d'énergie. Notre chiffrement de données étant symétrique, le coût en temps consommé et en énergie est équivalent aux méthodes symétriques. Notons que le schéma d'*Oliveira et al.* (TinyPBC) [95] correspond à l'implémentation la plus rapide (utilisation de la bibliothèque RELIC [123]) basée sur le couplage pour créer une clé commune partagée entre deux nœuds. L'exécution de cette opération sur le processeur MSP430 prend seulement 1,27 secondes. Notre implémentation basée sur la bibliothèque TinyECC permet de créer une clé commune en utilisant DHWI en 3,2 secondes. En revanche, nous utilisons cette opération dans notre proposition pour calculer uniquement la clé DH entre un nœud quelconque du réseau et la station de base. Tandis que TinyPBC utilise cette opération entre deux nœuds voulant établir un lien sécurisé. Ainsi, nous estimons que globalement notre méthode est équivalente en termes d'utilisation de ressources

et nous rappelons qu'elle est supérieure à TinyPBC (voir Tableau III.1).en termes de résistance contre les attaques.

IV.4 Conclusion

Nous avons présenté dans ce chapitre une implémentation et l'évaluation correspondante de notre proposition sur des cartes TelosB. Les résultats ont montré que notre proposition, basée sur l'utilisation de systèmes asymétriques et symétriques, est réalisable sur les deux topologies plates et hiérarchiques. La comparaison de nos résultats avec d'autres méthodes en termes d'utilisation de ressources de nœuds montrent que notre proposition peut être classée « au milieu ». En effet, notre proposition représente un compromis entre un bon niveau de sécurité et une consommation acceptable et moyenne de l'énergie des nœuds d'un RCSF. Les phases de distribution de clés, de découverte du voisinage, et de renouvellement et de révocation de clés nécessitent beaucoup de temps d'exécution et d'énergie. Dans notre solution, la phase de distribution n'est réalisée qu'une seule fois juste après le déploiement, tandis que les phases de découverte du voisinage et de renouvellement et révocation de clés sont effectuées sur demande. Dans le cas de capture de nœuds par exemple, le coût de ces phases augmente avec le nombre de nœuds capturés.

Les résultats obtenus à partir d'une implémentation représentent des indicateurs dont les valeurs sont susceptibles de varier en fonction de la technique de programmation et de compromis faits entre empreinte mémoire et rapidité d'exécution par exemple. Néanmoins, ce travail nous permet d'estimer un ordre de grandeur des délais et de la taille mémoire qu'il est nécessaire d'associer à chaque opération élémentaire d'une architecture globale sécurisée.

Conclusion générale

Les RCSF sont utilisés afin de faire un lien de communication entre les données générées par un processus d'application donné et leurs traitements et stockages dans des serveurs appropriés. Les besoins en sécurité relatifs à ce type de réseaux diffèrent d'un domaine applicatif à un autre et restent tout de même à ce jour un véritable challenge à la fois théorique et technologique. Nous avons proposé et évalué une nouvelle architecture dynamique de sécurité pour les RCSF. Elle permet de garantir et de maintenir la sécurité des communications durant toute la durée de vie du réseau. Nos comparaisons de résultats avec d'autres méthodes en termes d'utilisation de ressources de nœuds et résistances aux attaques montrent que notre proposition peut être classée au milieu. Elle présente un compromis entre un bon niveau de sécurité et une consommation moyenne de l'énergie des nœuds du RCSF.

Après avoir proposé une phase d'association au réseau qui garantit l'authentification des nouveaux nœuds et la délivrance des clés symétriques pour le chiffrement de données, nous l'avons complété avec deux phases. Une première qui garantit à un nœud la création de plusieurs chemins sécurisés vers la station de base, et une deuxième qui assure le renouvellement et la révocation de clés distribuées. Afin de prouver la faisabilité de notre proposition, nous l'avons implémentée sur des cartes TelosB. Les résultats issus des mesures réelles sur ces cartes sont satisfaisants et comparables à des méthodes existantes. Nous avons décortiqué des différents types d'opérations cryptographiques afin de mieux comprendre les sources de certains surcoûts. Ces résultats diffèrent bien sûr d'un matériel à un autre selon les capacités de calcul du système utilisé et de la technique de programmation utilisée. Malgré tout, les chiffres montrent que la sécurisation des communications dans un RCSF a un coût (temps d'exécution et empreinte mémoire) non négligeable qu'il faut prendre en compte. Les opérations de cryptographie rendent le réseau plus lent, ce fait doit intervenir dans le choix du niveau de sécurité à associer à une application donnée avant même la conception des protocoles de réseaux.

Perspectives

L'implémentation, les résultats et les comparaisons effectués dans le cadre de cette thèse nous ont incités à envisager des améliorations dans le but d'augmenter le niveau de sécurité et la dynamicité de notre proposition. Nous présentons dans la suite ces pistes d'amélioration.

V.1 Clés symétriques entre des nœuds non voisins

Nous avons supposé dans les deux topologies auxquelles nous avons dédié notre proposition que la station de base est le seul nœud destinataire des données récoltées des capteurs (le seul puits). Notre phase de découverte de voisins *NeighbDic* permet à un nœud d'établir des clés symétriques avec ses voisins afin de chiffrer les données destinées à la station de base. Avec l'existence d'autres puits de données, nous pouvons améliorer cette phase pour permettre à un nœud d'établir des clés symétriques avec un nœud non voisin. Ainsi, la découverte d'un nœud non voisin pourra être réalisée en passant par une phase de récupération de clé publique de ce dernier auprès de la station de base, avant d'essayer d'établir un lien multi-sauts sécurisé de bout-en-bout jusqu'au puits choisi. Ce travail évoque le *cross-layering* entre le protocole de routage au niveau de la couche réseau et l'établissement de lien sécurisé au niveau de la couche application qui englobe généralement les couches session et présentation du modèle OSI (comme c'est le cas pour le standard ZigBee).

V.2 L'utilisation de saut de fréquences

Nous envisageons d'améliorer notre proposition en ajoutant une phase basée sur un échange secret d'une séquence de saut de fréquences entre les nœuds du RCSF [124] [125]. Cette phase permet aux nœuds d'être capables de changer de fréquences de communication d'une transmission à une autre. Les bénéfices de cette technique dans les RCSF sont cités dans [126] et [127]. Cette technique nous incite à adopter un protocole MAC synchrone qui permet d'établir des rendez-vous sur les bonnes fréquences entre les nœuds voisins afin de pouvoir échanger des données.

Dans la Figure V.1, nous présentons un simple exemple de saut de fréquences. Au début de la communication, un nœud A envoie une séquence de fréquences parmi les fréquences utilisables à un nœud B. À la réception, B stocke cette séquence et commence à écouter sur les fréquences de cette séquence.

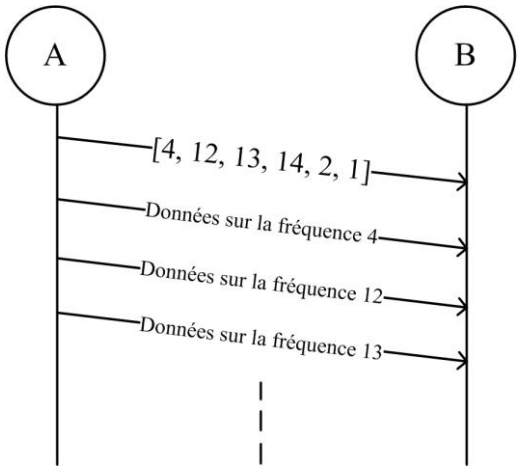


Figure V.1 : Communication d'une séquence de saut de fréquences.

Avant d'envoyer la séquence de saut, les nœuds A et B peuvent scanner leur environnement local. Il s'agit de déduire par écoute ou par des échanges de trames sondes quels sont les canaux : les fréquences, libres et/ou présentant des caractéristiques acceptables pour une transmission (rapport signal sur bruit par exemple). Lorsque A et B échangent leurs séquences formées, ils peuvent tenir compte des fréquences communes. Par conséquent, les futures communications peuvent être réalisées sur les liens les plus robustes. Les séquences de saut de fréquences peuvent être chiffrées et signées avant d'être envoyées pour assurer leur confidentialité et leur intégrité. (voir annexe VI.2)

Annexes

VI.1 Génération de clés RSA

Avant de décrire la génération des clés de la méthode RSA, nous devons fournir un complément d'arithmétique modulaire et de théorie des nombres. Nous présentons dans une première partie l'algorithme d'Euclide pour générer les clés. Ensuite, nous présentons dans la deuxième partie un théorème et des corollaires utiles qui aident à implémenter cette méthode.

VI.1.1 Algorithme d'Euclide étendu

L'algorithme d'Euclide étendu est utilisé pour calculer le plus grand commun diviseur de deux entiers positifs a et b. L'algorithme fixe r_0 à a et r_1 à b puis effectue la suite de divisions suivante :

$$\begin{aligned} r_0 &= q_1 r_1 + r_2, && 0 < r_2 < r_1 \\ r_1 &= q_2 r_2 + r_3, && 0 < r_3 < r_2 \\ &\dots \\ r_{m-1} &= q_m r_m, && 0 < r_m < r_{m-1} \end{aligned}$$

Puis, il définit les deux suites de nombres $t_0, t_1, \dots, t_m$ et $s_0, s_1, \dots, s_m$ selon la relation de récurrence suivante :

$$t_j = \begin{cases} 0 & si\ j = 0 \\ 1 & si\ j = 1 \\ t_{j-2} - q_{j-1} t_{j-1}\ si\ j \geq 2 \end{cases}$$

$$t_j = \begin{cases} 1 & si\ j = 0 \\ 0 & si\ j = 1 \\ s_{j-2} - q_{j-1} s_{j-1}\ si\ j \geq 2 \end{cases}$$

D'après [128], nous pouvons démontrer par induction que pour $0 \leq j \leq m$, nous avons $r_j = s_j r_0 + t_j r_1$.

Algorithme: Euclide étendu (a, b)

Entrées : a, b

Sorties : r, s, t

$a_0 \leftarrow a$
$b_0 \leftarrow b$
$t_0 \leftarrow 0$
$t \leftarrow 1$
$s_0 \leftarrow 1$
$s \leftarrow 0$
$q \leftarrow \left\lfloor \frac{a_0}{b_0} \right\rfloor$
$r \leftarrow a_0 - q b_0$

Tant que $r > 0$ faire
    $temp \leftarrow t_0 - qt$
    $t_0 \leftarrow t$
    $t \leftarrow temp$
    $temp \leftarrow s_0 - qs$
    $s_0 \leftarrow s$
    $s \leftarrow temp$
    $a_0 \leftarrow b_0$
    $b_0 \leftarrow r$
    $q \leftarrow \left\lfloor \frac{a_0}{b_0} \right\rfloor$
    $r \leftarrow a_0 - q b_0$
fin tant que

$r \leftarrow b_0$
retourner (r, s, t)

VI.1.2 Théorème et corollaires utiles

- Théorème

Si G est un groupe multiplicatif d'ordre n, et si $g \in G$, alors l'ordre de g divise n.

- Corollaire1

Si $b \in$ (Z/nZ)*, alors $b^{\varphi(n)} \equiv 1\ (mod\ n)$.

- Corollaire2

Vérifions que le chiffrement et le déchiffrement sont bien des opérations réciproques.

$$ab \equiv 1 \left(mod\ \varphi(n)\right),$$

Nous avons :

$$ab = t\varphi(n) + 1$$

Pour tout entier $t \geq 1$. Soit $x \in$ (Z/nZ)*. Nous avons :

$$(x^b)^a \equiv x^{t\varphi(n)+1}\ (mod\ n)$$

$$\equiv (x^{\varphi(n)})^t x\ (mod\ n)$$

$$\equiv 1^t x\ (mod\ n)$$

$$\equiv x\ (mod\ n)$$

VI.1.3 Choix des deux nombres premiers p et q pour RSA

Les exigences à prendre sur *p* et *q* afin d'avoir un bon niveau de sécurité sont les suivantes :

- Il faut que p et q soient deux nombres premiers suffisamment grands.
- p et q doivent être de même taille (en bits) afin d'éviter la factorisation de n (e*lliptic curve factoring*).
- Il faut que la différence p - q soit grande. Car sinon, l'adversaire considère l'équation $p = q + k$ avec k un petit entier. Et, comme il connaît $n = p \times q$, il est ramené à résoudre une équation de second degré pour chaque valeur de k.
- Il faut que p et q soient deux nombres premiers forts. Un nombre premier est dit fort s'il vérifie les conditions suivantes :
    a) $p - 1$ a un grand facteur premier r.
    b) $p + 1$ a un grand facteur premier s.
    c) $r - 1$ a un grand facteur premier t.

La condition a) permet de se prémunir contre la factorisation de p par l'algorithme P-1 de Pollard. La condition b) permet de se prémunir contre la factorisation de p par l'algorithme de P+1, attribué à Williams. Enfin, la condition c) permet de se prémunir contre les attaques cycliques.

Tests de primalité

- Définition

Un *algorithme de Monte Carlo positif* est un algorithme probabiliste qui résout un problème de décision tel que toute réponse positive (« oui ») est toujours correcte mais pour lequel une réponse négative (« non ») peut être incorrecte. On dit qu'un algorithme de Monte

Carlo positif a une probabilité d'erreur de ϵ si, pour chaque question dont la réponse devrait être positive, l'algorithme donne une réponse négative avec probabilité au plus ε.

- Miller-Rabin

Dans la mise en œuvre du chiffrement RSA, il faut engendrer de grands nombres premiers aléatoires. En pratique, on fabrique des nombres aléatoires et on teste leur primalité jusqu'à l'obtention d'un nombre premier. Supposons qu'on utilise l'algorithme de Miller-Rabin comme algorithme de test de primalité. L'algorithme de Miller-Rabin est un algorithme de Monte Carlo positif pour le problème de factorisation. L'algorithme ne peut pas répondre « n est composé » si n est premier. Tandis que l'algorithme peut répondre « n est premier » si n n'est pas premier avec une probabilité d'erreur au plus $\frac{1}{4}$.

Algorithme: de Miller-Rabin (n)

Ecrire n-1=$2^k m$, où m est impair

- **répéter t fois :**
- choisir un entier aléatoire $a, 1 \leq a \leq n-1$
- $b \leftarrow a^m \ (mod\ n)$
- si $b \equiv 1 (mod\ n)$
- alors retourner (« n est premier »)
- pour $i \leftarrow 0\ jusqu'$à $k-1$
    - si $b \equiv 1\ (mod\ n)$
    - alors retourner (« n est premier »)
    - sinon $b \leftarrow b^2\ (mod\ n)$
- fin pour
- **fin répéter**

Retourner (« n est composé »)

Efficacité

Comme tous les tests de primalité probabilistes, il existe des valeurs de n qui produiront de manière répétée des *menteurs*, qui indiqueront que n est premier alors qu'il est composé. Ces valeurs sont appelées **pseudo premières**. Dans l'algorithme de Miller-Rabin, plus on teste de valeurs de a, meilleure est la précision du test. En pratique, il suffit de tester 6

valeurs de a pour garantir que la probabilité d'erreur soit inférieur à 2^{-80}. Ce qui nous assure que ce test peut être utilisé de manière sûre dans les applications cryptologiques.

VI.1.4 Génération de clés

La génération des clés consiste à construire les deux couples « (n, e) : clé publique et (n, d) : clé privée ». Comme déjà vu, l'algorithme d'Euclide étendu permet de calculer le plus grand commun diviseur de deux entiers positifs *a* et *b*. Dans ce cas, on pourra utiliser cet algorithme afin de calculer les deux clés (clé publique, clé privée).

- Choix de la clé publique e

Il faut choisir e, tel que e est premier avec φ (n). Plusieurs choix ont été décrits [129]:

  - Choix aléatoire de e

La génération aléatoire de e n'est pas souhaitable, car elle ne gère et ne limite pas la complexité de l'algorithme d'exponentiation modulaire (le nombre de 1's peut être significatif en sa représentation binaire).

  - Choix de $e = 3$

L'exposant de chiffrement e=3 est couramment utilisé en pratique, car le chiffrement est rapide (l'algorithme d'exponentiation modulaire ne comporte que deux opérations modulaires).

Malgré cet avantage, l'exposant $e = 3$ n'est plus utilisé dans certaines applications afin de se prémunir contre l'attaque basée sur l'algorithme de Gauss.

  - Choix de $e = 2^{16} + 1$

Comme déjà vu, le choix de $e = 3$ n'est plus utilisé dans certaines applications qui demandent un niveau de sécurité potentiellement élevé. Pour cela, il faut choisir d'autres grandes valeurs de *e* tout en respectant le compromis entre la valeur de *e* et le temps de chiffrement (qui dépend du nombre d'opérations à effectuer selon l'algorithme d'exponentiation modulaire).

Nous avons utilisé dans ce projet, la valeur de chiffrement $e = 2^{16} + 1$ (si e n'est pas premier avec φ(n), on incrémente e de 2 et on teste de nouveau) [129]. C'est une valeur robuste (ne comporte que deux 1's en sa représentation binaire) et elle permet de se prémunir contre l'attaque basée sur l'algorithme de Gauss.

- Choix de la clé privée *d*

L'exposant de déchiffrement *d* est obtenu en utilisant l'algorithme d'Euclide étendu.

En 1990, M. Wiener a publié une attaque contre les clés privées trop petites. Plus précisément, son attaque permet de calculer la clé privé *d* uniquement en connaissant la clé publique (n, e) (et sans factoriser n), en $\log n$ étapes (chaque étape ayant un coût polynomial en nombre d'opérations sur les bits), lorsque $d < \frac{n^{1/4}}{3}$.

Pour cela, il faut que $d > \frac{n^{1/4}}{3}$ afin de se prémunir contre l'attaque de Wiener [130].

Exemple de format en hexadécimales des clés générées par la méthode RSA :

n	**e ou d**
D8052888E72D251E9AD8E09	C31DADA61DA708D0ADF77F1

VI.2 Saut de fréquence

Jones et al. [74] a combiné la cryptographie symétrique avec l'utilisation de saut de fréquences comme outil de sécurité. Avant le déploiement, tout le réseau partage une séquence secrète de saut de fréquences FHS *(Frequency-Hopping Secret)* utilisée dans la phase initiale après le déploiement. En utilisant ce secret FHS, la station de base envoie aux nœuds de chaque secteur i, $(0 < i < n$ où *n* est le nombre total des secteurs) la clé symétrique du secteur, WK_i. (voir Figure II.21). En ce moment, la station de base pourrait envoyer de nouvelles séquences secrètes de saut de fréquences propres à chaque secteur $WFHS_i$ *(Wedge Frequency-Hopping Secret).* Chaque nœud source du secteur utilise la clé symétrique du secteur WK_i combinée avec l'utilisation de saut de fréquences secrètes $WFHS_i$ dans le but de transmettre à ses voisins son porte-clés de clés symétriques en toute sécurité. À leur tour, les voisins envoient de la même manière leur porte-clés à leurs voisins jusqu'à la station de base. Le nœud source crée ensuite la clé PK_j du chemin de confiance *j* établi avec la station de base. Il utilise PK_j pour transmettre de nouvelles séquences de sauts de fréquences ($PFHS_j$) dédiées au chemin *j*. Cette méthode de sauts sert à renforcer la sécurité de la communication et rend le réseau plus robuste mais elle n'est pas suffisante d'être utilisée toute seule.

Les auteurs de [131] ont proposé un mécanisme basé sur le saut de fréquences afin de lutter principalement contre les interférences et le *jamming*. La méthode proposée s'appelle

ACH *(Adaptative Channel Hopping).* L'idée est que le parent d'un groupe génère périodiquement une séquence de sauts de fréquences et la transmet à ces fils. Les auteurs supposent que les parents connaissent leurs fils. La coordination entre les deux est gérée par deux types de messages : (i) CHC *(Channel Hopping Command)* : une commande utilisée par le père annonçant à ses fils l'existence d'un canal opérationnel, l'envoi des informations liées au changement d'un canal ou la confirmation de la disponibilité d'un canal. (ii) CHR *(Channel Hopping Reply)* : commande utilisée par un fils pour demander un saut de canal ou répondre à une commande pour envoyer les canaux disponibles de son groupe. Dans la Figure VI.1, nous présentons le mécanisme de changement de fréquence selon ACH. Au début, le parent de groupe *P* diffuse sur la fréquence courante f_c une requête de changement de canal f_c à f_n à ses fils (F_i). Une fois les requêtes reçues, ils envoient à leur parent une confirmation en utilisant la nouvelle fréquence f_n pour confirmer que cette fréquence est disponible pour eux. Si *P* reçoit des confirmations de tous ses fils, il envoie une confirmation à tout le monde confirmant l'utilisation courante de f_n. Sinon, il refait la requête de demande CHC de nouveau jusqu'à sa réussite.

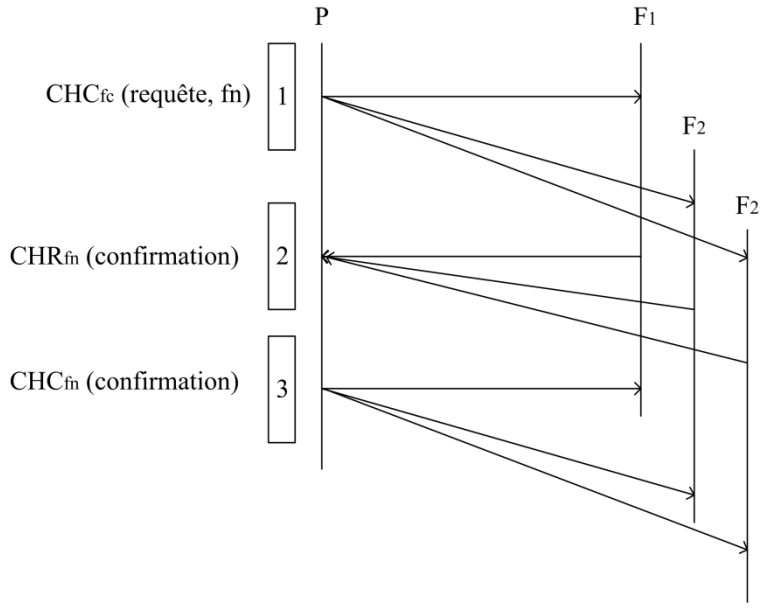


Figure VI.1 : Le mécanisme de saut de fréquences de ACH. Le mécanisme se réalise en trois étapes.

Nous comptons utiliser cette technique dans notre proposition afin de la rendre plus robuste contre des attaques spécifiques aux RCSF, comme le *Jamming*. En effet, l'utilisation des moyens cryptographiques pour sécuriser les échanges dans les RSCF n'est pas suffisante pour se défendre contre toutes les attaques. Imaginons par exemple, que les paquets échangés entre deux nœuds sont bien chiffrés et les moyens d'authentification ont été assurés, mais les

nœuds ne peuvent pas communiquer à cause d'une attaque comme le *Jamming* où la plupart des canaux sont rendus indisponibles. Dans ce cas, tous les efforts qui ont été fait pour assurer la confidentialité, l'intégration des données et l'authentification sont perdus. Le saut de fréquence est la solution idéale pour ce genre d'attaque. Notre phase se base sur un échange secret d'une séquence de fréquence entre chaque deux nœud du réseau. Une négociation de cette séquence est possible. Par exemple, un nœud A pourra tester les fréquences au début du déploiement afin de choisir une liste de bonnes fréquences d'envoi et de réception. Ensuite, il envoie cette liste à un autre nœud B. Ce dernier, possédant sa propre liste, essaye de trouver le plus grand nombre de fréquences communes afin de construire la séquence de saut de fréquence. Cette séquence sera envoyée secrètement à A. Elle pourra être chiffrée avec la clé de session déjà établie dans la phase *JoinNet* ou *NeighbDisc*. Dans ce qui suit, Nous montrons la possibilité d'appliquer le saut de fréquences sur une topologie hiérarchique comme MaCARI. Afin d'établir un temps de synchronisation qui permet aux nœuds de changer d'une fréquence à une autre, nous pourrons utiliser un mécanisme de synchronisation basé sur la propagation de trames de Beacon (comme celui utilisé par MaCARI). Le temps est divisé en cycles. Le premier cycle est consacré à la propagation des trames de Beacon d'une façon hiérarchique en allant de la station de base à travers les routeurs le long de la topologie. La propagation hiérarchique est nécessaire afin d'éviter les collisions entre les trames de Beacon. Une trame de Beacon contient des informations comme la segmentation de temps permettant aux nœuds du réseau de savoir le moment de mise en veille pour économiser de l'énergie et le moment de réveil pour envoyer ou recevoir des données. Comme le montre la Figure VI.2, la période $[T_0, T_1]$ est dédiée à la synchronisation. Dans la période $[T_1, T_2]$, la station de base attribue à chaque routeur un slot de temps lui permettant de communiquer avec son étoile. Les données collectées par les routeurs sont envoyées aux routeurs pères jusqu'à l'arrivée à leur destination finale. Afin de laisser des nouveaux nœuds rejoindre le réseau d'une façon dynamique, nous pourrons utiliser le CSMA/CA durant la période $[T_2, T_3]$. Les nouveaux nœuds envoient une requête aux routeurs les plus proches d'eux et attendent une réponse d'acceptation afin de rejoindre le réseau. Ces nouvelles demandes sont similaires aux demandes des premiers nœuds du réseau durant la phase *JoinNet*. Finalement, la période $[T_3, T_0]$ est considérée comme une période inactive durant laquelle les nœuds peuvent se mettre en veille pour économiser leurs énergies.

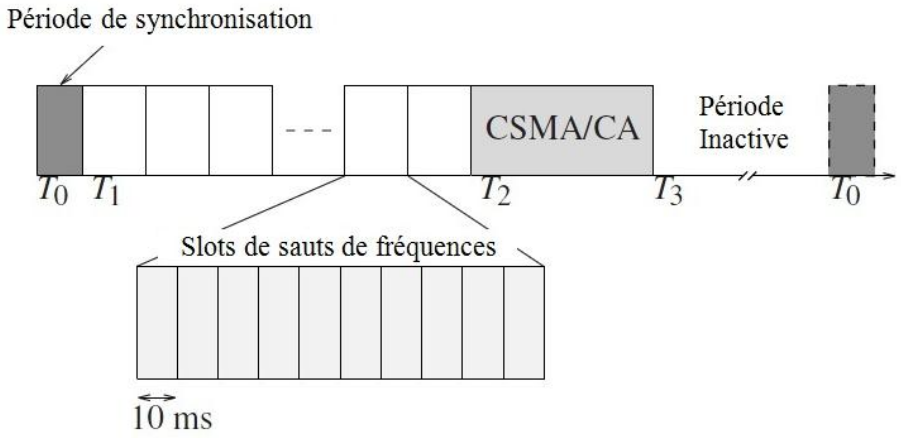


Figure VI.2 : La segmentation de temps pour le mécanisme de saut de fréquence.

Listes des publications internationales

1. **I. Mansour**, G. Chalhoub, et M. Misson, « Security architecture for multi-hop wireless sensor networks », in *Security for Multihop Wireless Networks*, USA: CRC Press Book, 2013. (à paraître en décembre 2013)

2. **I. Mansour**, G. Chalhoub, et B. Bakhache, « Evaluation of a Fast Symmetric Cryptographic Algorithm Based on the Chaos Theory for Wireless Sensor Networks », in *IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, Liverpool, UK, 2012, p. 913‑919.

3. **I. Mansour** et G. Chalhoub, « Evaluation of different cryptographic algorithms on wireless sensor network nodes », in *International Conference on Wireless Communications in Unusual and Confined Areas (ICWCUCA)*, Clermont-FD, France, 2012, p. 1‑6.

4. **I. Mansour**, G. Chalhoub, et A. Quilliot, « Security architecture for wireless sensor networks using frequency hopping and public key management », in *IEEE International Conference on Networking, Sensing and Control (ICNSC)*, Delft, Pays-Bas, 2011, p. 526‑531.

5. **I. Mansour**, G. Chalhoub, et M. Misson, « Energy-efficient Security Protocol for Wireless Sensor Networks using Frequency Hopping and Permutation Ciphering », in *International Conference on Pervasive and Embedded Computing and Communication Systems (PECCS), Poster*, Algarve, Portugal, 2011, p. 277‑282.

Références

[1] REPLIC, « Équipe Réseaux et Protocoles du LIMOS ». [En ligne]. Disponible sur: http://sancy.univ-bpclermont.fr/. [Consulté le: 19-avr-2013].

[2] LIMOS, « Laboratoire d'Informatique, de Modélisation et d'Optimisation des Systèmes ». [En ligne]. Disponible sur: http://limos.isima.fr/. [Consulté le: 19-avr-2013].

[3] UBP, « Université Blaise Pascal de Clermont-Ferrand ». [En ligne]. Disponible sur: http://www.univ-bpclermont.fr/. [Consulté le: 19-avr-2013].

[4] FEDER, « Fonds européen de développement régional ». [En ligne]. Disponible sur: http://www.europe-en-auvergne.eu/. [Consulté le: 19-avr-2013].

[5] ZigBee Alliance, « The ZigBee Alliance web site ». [En ligne]. Disponible sur: http://www.zigbee.org/.

[6] HART Communication Foundation Std., « HART field communication protocol specifications », Tech. Rep., 2008.

[7] International Society of Automation Std.,, « ISA100.11a : 2009 wireless systems for industrial automation : Process control and related applications », Draft standard, in preparation, 2009.

[8] K. Romer et F. Mattern, « The design space of wireless sensor networks », *IEEE Wireless Communications*, vol. 11, n° 6, p. 54‑61, 2004.

[9] DARPA, « Defense Advanced Research Projects Agency ». [En ligne]. Disponible sur: http://www.darpa.mil/. [Consulté le: 04-mai-2013].

[10] G. Simon, M. Maróti, Á. Lédeczi, G. Balogh, B. Kusy, A. Nádas, G. Pap, J. Sallai, et K. Frampton, « Sensor network-based countersniper system », in *Proceedings of the 2nd international conference on Embedded networked sensor systems*, New York, USA, 2004, p. 1–12.

[11] J. K. Hart et K. Martinez, « Environmental Sensor Networks: A revolution in the earth system science? », *Earth-Science Reviews*, vol. 78, n° 3‑4, p. 177‑191, oct. 2006.

[12] V. C. Gungor et G. P. Hancke, « Industrial Wireless Sensor Networks: Challenges, Design Principles, and Technical Approaches », *IEEE Transactions on Industrial Electronics*, vol. 56, n° 10, p. 4258‑4265, 2009.

[13] J. López et J. Zhou, *Wireless Sensor Network Security*. IOS Press, 2008.

[14] C. Allen et T. Dierks, « The TLS Protocol Version 1.0 ». [En ligne]. Disponible sur: https://tools.ietf.org/html/rfc2246. [Consulté le: 18-sept-2012].

[15] K. Seo et S. Kent, « Security Architecture for the Internet Protocol ». [En ligne]. Disponible sur: http://tools.ietf.org/html/rfc4301. [Consulté le: 19-sept-2012].

[16] S. Fouladgar, B. Mainaud, K. Masmoudi, et H. Afifi, « Tiny 3-TLS: A Trust Delegation Protocol for Wireless Sensor Networks », in *Security and Privacy in Ad-Hoc and Sensor Networks*, vol. 4357, 2006, p. 32‑42.

[17] R. Mzid, M. Boujelben, H. Youssef, et M. Abid, « Adapting TLS handshake protocol for heterogenous IP-based WSN using identity based cryptography », in *International Conference on Communication in Wireless Environments and Ubiquitous Systems: New Challenges (ICWUS)*, 2010, p. 1 ‑8.

[18] T. Kavitha1 et D. Sridharan, « Security Vulnerabilities In Wireless Sensor Networks: A Survey », *Journal of Information Assurance and Security*, vol. 5, p. 31–44, 2010.

[19] Y. Wang, G. Attebury, et B. Ramamurthy, « A Survey of Security Issues In Wireless Sensor Networks », *CSE Journal Articles*, janv. 2006.

[20] Al Murat et Yoshigoe Kenji, « Security and Attacks in Wireless Sensor Networks », in *Network Security, Administration and Management: Advancing Technology and Practice*, 2011, p. 183‑216.

[21] D. G. Padmavathi et M. D. Shanmugapriya, « A Survey of Attacks, Security Mechanisms and Challenges in Wireless Sensor Networks », *International Journal of Computer Science and Information Security*, vol. 4, sept. 2009.

[22] W. Stallings, *Cryptography and Network Security: Principles and Practice*. Prentice Hall, 2006.

[23] A. S. K. Pathan et C. S. Hong, « Security Attacks and Challenges in Wireless Sensor Networks », in *Encyclopedia on ad hoc and ubiquitous computing: theory and design of wireless ad hoc, sensor, and mesh networks*, vol. 16, 2009.

[24] A. Uluagac, C. Lee, R. Beyah, et J. Copeland, « Designing Secure Protocols for Wireless Sensor Networks », in *Wireless Algorithms, Systems, and Applications*, vol. 5258, Y. Li, D. Huynh, S. Das, et D.-Z. Du, Éd. 2008, p. 503‑514.

[25] S. K. Jain et K. Garg, « A Hybrid Model of Defense Techniques against Base Station Jamming Attack in Wireless Sensor Networks », in *First International Conference on Computational Intelligence, Communication Systems and Networks.*, 2009, p. 102 ‑107.

[26] G. Alnifie et R. Simon, « A multi-channel defense against jamming attacks in wireless sensor networks », in *Proceedings of the 3rd ACM workshop on QoS and security for wireless and mobile networks*, New York, USA, 2007, p. 95–104.

[27] W. Xu, W. Trappe, Y. Zhang, et T. Wood, « The feasibility of launching and detecting jamming attacks in wireless networks », in *Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*, New York, USA, 2005, p. 46–57.

[28] N. Ahmed, S. S. Kanhere, et S. Jha, « The holes problem in wireless sensor networks: a survey », *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 9, n° 2, p. 4–18, 2005.

[29] B. Karp et H. T. Kung, « GPSR: greedy perimeter stateless routing for wireless networks », in *Proceedings of the 6th annual international conference on Mobile computing and networking*, New York, NY, USA, 2000, p. 243–254.

[30] Y. Yu, R. Govidan, et D. Estrin, « Geographical and energy aware routing: A recursive data dissemination protocol for wireless sensor networks », UCLA Computer Science Department Technical Report, 2001.

[31] J. Douceur, « The Sybil Attack », in *Peer-to-Peer Systems*, vol. 2429, P. Druschel, F. Kaashoek, et A. Rowstron, Éd. 2002, p. 251‑260.

[32] Q. Zhang, P. Wang, D. S. Reeves, et P. Ning, « Defending against Sybil attacks in sensor networks », in *Distributed Computing Systems Workshops, 2005. 25th IEEE International Conference on*, 2005, p. 185 ‑ 191.

[33] C. Karlof, N. Sastry, et D. Wagner, « TinySec: a link layer security architecture for wireless sensor networks », in *Proceedings of the 2nd international conference on Embedded networked sensor systems*, New York, NY, USA, 2004, p. 162–175.

[34] J. Wang, G. Yang, Y. Sun, et S. Chen, « Sybil Attack Detection Based on RSSI for Wireless Sensor Network », in *Wireless Communications, Networking and Mobile Computing, 2007. WiCom 2007. International Conference on*, 2007, p. 2684 ‑2687.

[35] D. Mukhopadhyay et I. Saha, « Location Verification Based Defense Against Sybil Attack in Sensor Networks », in *Distributed Computing and Networking*, vol. 4308, S. Chaudhuri, S. Das, H. Paul, et S. Tirthapura, Éd. 2006, p. 509‑521.

[36] H. Yu, M. Kaminsky, P. B. Gibbons, et A. Flaxman, « SybilGuard: defending against sybil attacks via social networks », in *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, New York, NY, USA, 2006, p. 267–278.

[37] C. P. Pfleeger et S. L. Pfleeger, *Security in computing*. Prentice Hall Professional, 2003.

[38] C. Hartung, J. Balasalle, et R. Han, « Node compromise in sensor networks: The need for secure systems », *University of Colorado at Boulder, Technical Report TR-CU-CS-990-05*, 2005.

[39] S. Zhu, S. Setia, et S. Jajodia, « LEAP+: Efficient security mechanisms for large-scale distributed sensor networks », *ACM Trans. Sen. Netw.*, vol. 2, n° 4, p. 500–528, nov. 2006.

[40] D. Liu, P. Ning, et R. Li, « Establishing pairwise keys in distributed sensor networks », *ACM Trans. Inf. Syst. Secur.*, vol. 8, n° 1, p. 41–77, févr. 2005.

[41] D. Martynov, J. Roman, S. Vaidya, et H. Fu, « Design and implementation of an intrusion detection system for wireless sensor networks », in *IEEE International Conference on Electro/Information Technology*, 2007, p. 507‑512.

[42] B. Sun, L. Osborne, Y. Xiao, et S. Guizani, « Intrusion detection techniques in mobile ad hoc and wireless sensor networks », *IEEE Wireless Communications*, vol. 14, n° 5, p. 56 ‑ 63, oct. 2007.

[43] G. CHASSÉ, *Cryptographie*. Ed. Techniques Ingénieur, 2000.

[44] AES, « Advanced Encryption Standard », U.S. DoC/NIST, 2001.

[45] A. Baranovsky et D. Daems, « Design of One-Dimensional chaotic maps with prescribed statistical properties », *International Journal of Bifurcation and Chaos*, vol. 05, n° 06, p. 1585‑1598, déc. 1995.

[46] S. Li, G. Chen, et X. Mou, « On the Dynamical Degradation of Digital Piecewise Linear Chaotic Maps », *International Journal of Bifurcation and Chaos*, vol. 15, p. 3119‑3151, 2005.

[47] B. Bakhache, J. Ghazal, et S. El Assad, « Enhancement of ZigBee and Wi-Fi security by a robust and fast chaotic algorithm », in *5th International Conference on Network and System Security (NSS)*, 2011, p. 300‑304.

[48] W. Diffie et M. E. Hellman, « New directions in cryptography », *IEEE Transactions on Information Theory*, vol. 22, p. 644–654, 1976.

[49] T. ElGamal, « A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms », in *Advances in Cryptology*, vol. 196, G. Blakley et D. Chaum, Éd. 1985, p. 10‑18.

[50] R. L. Rivest, A. Shamir, et L. Adleman, « A method for obtaining digital signatures and public-key cryptosystems », *Commun. ACM*, vol. 21, n° 2, p. 120–126, févr. 1978.

[51] J. D. Dixon, « Factorization and Primality Tests », *The American Mathematical Monthly*, vol. 91, n° 6, p. 333‑352, juin 1984.

[52] R. Merkle, « A Certified Digital Signature », in *Advances in Cryptology — CRYPTO' 89 Proceedings*, vol. 435, G. Brassard, Éd. Springer Berlin / Heidelberg, 1990, p. 218‑238.

[53] D. Eastlake et P. Jones, « US Secure Hash Algorithm 1 (SHA1) ». IETF Request for Comments 3174, sept-2001.

[54] R. Rivest, « The MD5 Message-Digest Algorithm ». IETF Request for Comments 1321, 1992.

[55] O. Mikle, « Practical attacks on digital signatures using MD5 message digest », *Cryprology ePrint Archive, report 2004*, vol. 356, 2004.

[56] X. Wang et H. Yu, « How to Break MD5 and Other Hash Functions », in *Advances in Cryptology – EUROCRYPT 2005*, vol. 3494, R. Cramer, Éd. 2005, p. 561‑561.

[57] V. Miller, « Use of Elliptic Curves in Cryptography », in *Advances in Cryptology - CRYPTO '85 Proceedings*, vol. 218, H. Williams, Éd. 1986, p. 417‑426.

[58] N. Koblitz, « Elliptic curve cryptosystems », *Mathematics of Computation*, vol. 48, p. 203–209, 1987.

[59] N. Koblitz, A. Menezes, et S. Vanstone, « The State of Elliptic Curve Cryptography », *Designs, Codes and Cryptography*, vol. 19, n° 2, p. 173‑193, 2000.

[60] Certicom Research, « Standards for efficient cryptography, SEC 1: Elliptic Curve Cryptography ». 2000.

[61] FIPS (Federal Information Processing Standards Publication), « Digital Signature Standard (DSS) », 1994. [En ligne]. Disponible sur: http://www.itl.nist.gov/fipspubs/fip186.htm. [Consulté le: 19-sept-2012].

[62] S. Farrell, R. Housley, et D.- Txt, *R.: RFC-3281. An Internet Attribute Certificate Profile for Authorization. The Internet Society*. 2002.

[63] A. J. Menezes, T. Okamoto, et S. A. Vanstone, « Reducing elliptic curve logarithms to logarithms in a finite field », *IEEE Transactions on Information Theory*, vol. 39, n° 5, p. 1639‑1646, sept. 1993.

[64] R. Sakai, K. Ohgishi, et M. Kasahara, « Cryptosystems based on pairing », in *SCIS 2000. Symposium on Cryptography and Information Security*, 2000, p. 26‑28.

[65] A. Joux, « A One Round Protocol for Tripartite Diffie–Hellman », in *Algorithmic Number Theory*, vol. 1838, W. Bosma, Éd. Berlin, Heidelberg, 2000, p. 385‑393.

[66] A. Shamir, « Identity-Based Cryptosystems and Signature Schemes », in *G.R. Blakley, D. Chaum (Eds.), Fourth Annual International Cryptology Conference*, 1984, vol. 196, p. 47‑53.

[67] Dan Boneh et A. Silverberg, « Applications of Multilinear Forms to Cryptography », in *Contemporary Mathematics*, vol. 324, 2003, p. 71‑90.

[68] D. Shaw, L. Donnerhacke, R. Thayer, H. Finney, et J. Callas, « OpenPGP Message Format ». [En ligne]. Disponible sur: http://tools.ietf.org/html/rfc4880. [Consulté le: 18-sept-2012].

[69] A. M. Hegland, E. Winjum, S. F. Mjolsnes, C. Rong, O. Kure, et P. Spilling, « A survey of key management in ad hoc networks », *IEEE Communications Surveys & Tutorials*, vol. 8, n° 3, p. 48–66, 2006.

[70] S. A. Camtepe et B. Yener, « Key Distribution Mechanisms for Wireless Sensor Networks: a Survey », 2005.

[71] S. Ruj, A. Nayak, et I. Stojmenovic, « Key Predistribution in Wireless Sensor Networks When Sensors Are Within Communication Range », in *Theoretical Aspects of Distributed Computing in Sensor Networks*, S. Nikoletseas et J. D. P. Rolim, Éd. Berlin, Heidelberg, 2011, p. 787‑832.

[72] L. Eschenauer et V. D. Gligor, « A key-management scheme for distributed sensor networks », in *Proceedings of the 9th ACM conference on Computer and communications security*, New York, NY, USA, 2002, p. 41–47.

[73] H. Chan, A. Perrig, et D. Song, « Random key predistribution schemes for sensor networks », in *2003 Symposium on Security and Privacy, 2003. Proceedings*, 2003, p. 197‑213.

[74] K. Jones, A. Wadaa, S. Oladu, L. Wilson, et M. Eltoweissy, « Towards a New Paradigm for Securing Wireless Sensor Networks », in *New Security Paradigms Workshop*, 2003.

[75] H. Chan et A. Perrig, « PIKE: peer intermediaries for key establishment in sensor networks », in *Proceedings IEEE INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, 2005, vol. 1, p. 524 ‑ 535 vol. 1.

[76] D. Liu, P. Ning, et W. Du, « Group-based key predistribution for wireless sensor networks », *ACM Trans. Sen. Netw.*, vol. 4, n° 2, p. 11:1–11:30, avr. 2008.

[77] S. A. Çamtepe et B. Yener, « Combinatorial Design of Key Distribution Mechanisms for Wireless Sensor Networks », in *Computer Security – ESORICS 2004*, P. Samarati, P. Ryan, D. Gollmann, et R. Molva, Éd. Springer Berlin Heidelberg, 2004, p. 293‑308.

[78] J. Lee et D. R. Stinson, « A combinatorial approach to key predistribution for distributed sensor networks », in *2005 IEEE Wireless Communications and Networking Conference*, 2005, vol. 2, p. 1200 ‑ 1205 Vol. 2.

[79] S. Ruj et B. Roy, « Key Predistribution Schemes Using Codes in Wireless Sensor Networks », in *Information Security and Cryptology*, M. Yung, P. Liu, et D. Lin, Éd. 2009, p. 275‑288.

[80] A. Perrig, R. Szewczyk, J. D. Tygar, V. Wen, et D. E. Culler, « SPINS: security protocols for sensor networks », *Wirel. Netw.*, vol. 8, n° 5, p. 521–534, sept. 2002.

[81] A. Perrig, R. Canetti, J. Tygar, et D. Song, « Efficient authentication and signing of multicast streams over lossy channels », in *IEEE Symposium on Security and Privacy*, 2000.

[82] R. L. Rivest, « The RC5 encryption algorithm », in *Fast Software Encryption*, B. Preneel, Éd. Springer Berlin Heidelberg, 1995, p. 86‑96.

[83] E. F. Brickell, « The SKIPJACK Algorithm », *Jul*, vol. 28, p. 1‑7, 1993.

[84] S. Zhu, S. Setia, et S. Jajodia, « LEAP+: Efficient security mechanisms for large-scale distributed sensor networks », *ACM Trans. Sen. Netw.*, vol. 2, n° 4, p. 500–528, nov. 2006.

[85] Zigbee, « Zigbee Specification », ZigBee Standards Organization, 053474r17, janv. 2008.

[86] G. Dini et I. M. Savino, « S2RP: a Secure and Scalable Rekeying Protocol for Wireless Sensor Networks », in *IEEE International Conference on Mobile Adhoc and Sensor Systems (MASS)*, 2006, p. 457 ‑466.

[87] B. Sinopoli, C. Sharp, L. Schenato, S. Schaffert, et S. S. Sastry, « Distributed control applications within sensor networks », *Proceedings of the IEEE*, vol. 91, n° 8, p. 1235 ‑ 1246, août 2003.

[88] E. Munivel et G. M. Ajit, « Efficient Public Key Infrastructure Implementation in Wireless Sensor Networks », in *International Conference on Wireless Communication and Sensor Computing*, 2010, p. 1 ‑6.

[89] R. Watro, D. Kong, S. Cuti, C. Gardiner, C. Lynn, et P. Kruus, « TinyPK: securing sensor networks with public key technology », in *Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks*, New York, NY, USA, 2004, p. 59–64.

[90] Zigbee, « ZigBee Smart Energy Specification », ZigBee Standards Organization, 075356r15, déc. 2008.

[91] Certicom Research, « Standards for Efficient Cryptography: SEC 1 (working draft) ver 1.7: Elliptic Curve Cryptography », nov. 2006.

[92] C. Lederer, R. Mader, M. Koschuch, J. Großschädl, A. Szekely, et S. Tillich, « Energy-Efficient Implementation of ECDH Key Exchange for Wireless Sensor Networks », in *Information Security Theory and Practice. Smart Devices, Pervasive Systems, and Ubiquitous Networks*, vol. 5746, O. Markowitch, A. Bilas, J.-H. Hoepman, C. Mitchell, et J.-J. Quisquater, Éd. 2009, p. 112‑127.

[93] D. Boneh et M. Franklin, « Identity-Based Encryption from the Weil Pairing », in *Advances in Cryptology — CRYPTO 2001*, J. Kilian, Éd. Springer Berlin Heidelberg, 2001, p. 213‑229.

[94] R. Sakai, K. Ohgishi, et M. Kasahara, « Cryptosystems based on pairing », in *Symposium on Cryptography and Information Security (SCIS'00)*, Japan, 2000, p. 26–28.

[95] L. B. Oliveira, D. F. Aranha, C. P. L. Gouvêa, M. Scott, D. F. Câmara, J. López, et R. Dahab, « TinyPBC: Pairings for authenticated identity-based non-interactive key distribution in sensor networks », *Computer Communications*, vol. 34, n° 3, p. 485‑493, mars 2011.

[96] Q. Jing, J. Hu, et Z. Chen, « C4W: An Energy Efficient Public Key Cryptosystem for Large-Scale Wireless Sensor Networks », in *2006 IEEE International Conference on Mobile Adhoc and Sensor Systems (MASS)*, 2006, p. 827‑832.

[97] Y.-S. Jeong et S.-H. Lee, « Hybrid Key Establishment Protocol Based on ECC for Wireless Sensor Network », in *Ubiquitous Intelligence and Computing*, vol. 4611, J. Indulska, J. Ma, L. Yang, T. Ungerer, et J. Cao, Éd. 2007, p. 1233‑1242.

[98] I. Mansour, G. Chalhoub, et M. Misson, « Security architecture for multi-hop wireless sensor networks », in *Security for Multihop Wireless Networks*, USA: CRC Press Book, 2013.

[99] K. Lauter, « The advantages of elliptic curve cryptography for wireless security », *IEEE Wireless Communications*, vol. 11, n° 1, p. 62‑67, 2004.

[100] G. Chalhoub, A. Guitton, et M. Misson, « MAC specifications for a WPAN allowing both energy saving and guaranted delay - Part A: MaCARI: a synchronized tree-based MAC protocol », in *IFIP WSAN*, 2008.

[101] T. Dang et C. Devic, « OCARI: Optimization of communication for Ad hoc reliable industrial networks », in *6th IEEE International Conference on Industrial Informatics, 2008. INDIN 2008*, 2008, p. 688‑693.

[102] TinyECC version 2.0 (02/03/2011), « A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks ». [En ligne]. Disponible sur: http://discovery.csc.ncsu.edu/software/TinyECC/. [Consulté le: 07-avr-2013].

[103] TinyOS Alliance, « TinyOS ». [En ligne]. Disponible sur: http://www.tinyos.net/. [Consulté le: 31-mars-2013].

[104] Crossbow, « TelosB datasheet ». Document Part Number: 6020-0094-01 Rev B.

[105] « Université de Berkeley ». [En ligne]. Disponible sur: http://www.berkeley.edu/. [Consulté le: 31-mars-2013].

[106] « The First European TinyOS Technology Exchange (ETTX 2009) », 10-févr-2009. [En ligne]. Disponible sur: https://sites.google.com/site/ettx2009/. [Consulté le: 31-mars-2013].

[107] Crossbow, « MicaZ datasheet ». Document Part Number: 6020-0060-04 Rev A.

[108] Crossbow Technology Inc., « Crosbow ». [En ligne]. Disponible sur: http://www.xbow.com/. [Consulté le: 31-mars-2013].

[109] J. L. Hill et D. E. Culler, « Mica: a wireless platform for deeply embedded networks », *IEEE Micro*, vol. 22, n° 6, p. 12‑24, 2002.

[110] J. Polastre, R. Szewczyk, et D. Culler, « Telos: enabling ultra-low power wireless research », in *Fourth International Symposium on Information Processing in Sensor Networks, 2005. IPSN 2005*, 2005, p. 364‑369.

[111] Moteiv Corporation, « Telos (Rev B) : PRELIMINARY Datasheet ». 12-mai-2004.

[112] I. Mansour et G. Chalhoub, « Evaluation of different cryptographic algorithms on wireless sensor network nodes », in *International Conference on Wireless Communications in Unusual and Confined Areas (ICWCUCA)*, Clermont-FD, France, 2012, p. 1‑6.

[113] I. Mansour, G. Chalhoub, et B. Bakhache, « Evaluation of a Fast Symmetric Cryptographic Algorithm Based on the Chaos Theory for Wireless Sensor Networks », in *IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, Liverpool, UK, 2012, p. 913‑919.

[114] NIST, « Recommendation for block cipher modes of operation, methods and techniques », U.S. DoC/NIST, Tech. Rep., déc. 2001.

[115] Zigbee, « Zigbee Specification », ZigBee Standards Organization, Zigbee Standard 053474r17, janv. 2008.

[116] Z.-H. Guan, F. Huang, et W. Guan, « Chaos-based image encryption algorithm », *Physics Letters A*, vol. 346, n° 1‑3, p. 153‑157, oct. 2005.

[117] Y. Wang, K.-W. Wong, X. Liao, T. Xiang, et G. Chen, « A chaos-based image encryption algorithm with variable control parameters », *Chaos, Solitons & Fractals*, vol. 41, n° 4, p. 1773‑1783, août 2009.

[118] Y. Mao, G. Chen, et S. Lian, « A novel fast image encryption scheme based on 3D chaotic baker maps », *International Journal of Bifurcation and Chaos*, vol. 14, n° 10, p. 3613‑3624, oct. 2004.

[119] G. Chen, Y. Mao, et C. K. Chui, « A symmetric image encryption scheme based on 3D chaotic cat maps », *Chaos, Solitons & Fractals*, vol. 21, n° 3, p. 749‑761, juill. 2004.

[120] IEEE 802.15, « Part 15.4: Wireless medium access control (MAC) and physical layer (PHY) specifications for low-rate wireless personal area networks (WPANs) », ANSI/IEEE, Standard 802.15.4 R2006, 2006.

[121] A. Liu et N. Ning, « TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks », in *7th International Conference on Information Processing in Sensor Networks*, 2008, p. 245‑256.

[122] Certicom Research, « Standards for efficient cryptography – SEC 2 : Recommended elliptic curve domain parameters », sept. 2000.

[123] D. F. Aranha et C. P. L. Gouv, « RELIC is an Efficient LIbrary for Cryptography ». [En ligne]. Disponible sur: http://code.google.com/p/relic-toolkit/.

[124] I. Mansour, G. Chalhoub, et M. Misson, « Energy-efficient Security Protocol for Wireless Sensor Networks using Frequency Hopping and Permutation Ciphering », in *International Conference on Pervasive and Embedded Computing and Communication Systems (PECCS), Poster*, Algarve, Portugal, 2011, p. 277‑282.

[125] I. Mansour, G. Chalhoub, et A. Quilliot, « Security architecture for wireless sensor networks using frequency hopping and public key management », in *IEEE International Conference on Networking, Sensing and Control (ICNSC)*, Delft, Pays-Bas, 2011, p. 526‑531.

[126] K. Pister et D. Lance, « TSMP: time synchronized mesh protocol », in *Distributed sensor networks*, 2006.

[127] T. Watteyne, A. Mehta, et K. Pister, « Reliability Through Frequency Diversity: Why Channel Hopping Makes Sense », in *PE-WASUN*, 2009.

[128] D. STINSON, *Cryptographie : Théorie et pratique*, 2ème édition. 1996.

[129] A. Menezes, P. Van Oorschot, et S. Vanstone, « Public-Key Encryption », in *Handbook of Applied Cryptography*, CRC Press, 1996.

[130] M. Wiener, B. N. R. Ltd, et O. Ottawa, « Cryptanalysis of short RSA secret exponents », in *Information Theory*, IEEE Transactions., vol. 36, 1990, p. 553‑558.

[131] S.-U. Yoon, R. Murawski, E. Ekici, S. Park, et Z. H. Mir, « Adaptive Channel Hopping for Interference Robust Wireless Sensor Networks », in *IEEE International Conference on Communications (ICC)*, 2010, p. 1‑5.

Résumé

Les réseaux de capteurs sans fil (RCSF) sont devenus un thème porteur aussi bien pour la recherche académique que pour les activités des services de R&D en raison de leur simplicité de déploiement et de leur potentiel applicatif dans des domaines très variés (militaire, environnemental, industriel). Un RCSF est composé d'un ensemble de nœuds devant être opérationnels et autonomes énergétiquement pour de longues périodes. De ce fait ils sont limités en capacité mémoire et de calcul, et contraint à exploiter une faible puissance de transmission, ce qui en limite leur portée et rend leur débit modeste.

Le besoin de sécuriser les communications dans un RCSF dépend de la criticité des données échangées pour l'application supportée. La solution doit reposer sur des échanges sûrs, confidentiels et fiables. Pour assurer la sécurisation des échanges, des techniques de cryptographie existent dans la littérature. Conçues à l'origine pour des réseaux informatiques majoritairement câblés, elles se basent généralement sur des algorithmes complexes et gourmands en ressource. Dans le cadre de cette thèse, nous avons proposé, implémenté et évalué une architecture sécurisée et dynamique adaptée aux communications des RCSF. Elle permet de garantir et de maintenir la sécurité des communications durant toute la durée de vie d'un réseau multi-saut. Nous avons utilisé et adapté des algorithmes standards de cryptographie, tels que AES-CTR et la suite d'algorithmes basée sur ECC, qui permettent à notre architecture de résister à la majorité d'attaques. Nous avons quantifié le surcoût en temps de calcul et en occupation mémoire de notre solution. Les résultats d'implémentation de notre proposition sont issus de mesures réelles faites sur une maquette réalisée à partir de cartes TelosB.

Mots-Clés : réseaux de capteurs sans fil, sécurité des communications, gestion de clés publiques, établissement de clé, ECC, ECDH, ECAES, AES-CTR.

Abstract

Wireless sensor networks (WSNs) have become an attractive topic for both academic research and the activity of R&D services due to their simple deployment and their potential of application in varied fields (military, environmental, industrial). A WSN is composed of a set of nodes that are supposed to operate and to be energetically autonomous for long durations. Thus, they are limited in memory and computing capacities, and constrained to function in a low-power transmission mode which limit their communication range and leave them with low data rates.

The need to secure communications in a WSN depends on the criticality of the exchanged data for the supported application. The solution must be based on safe, confidential and reliable exchanges. To ensure the security of exchanges, cryptographic techniques exist in the literature. Originally designed for mostly wired computer networks, they are usually based on complex and resource-consuming algorithms. In this thesis, we have proposed, implemented and evaluated a secure and dynamic architecture suitable for WSNs communications. It ensures and maintains secured communications throughout the lifetime of a multi-hop network. We have used and adapted standard cryptographic algorithms, such as AES-CTR and algorithms based on ECC cipher suites, which allow our architecture to resist against most attacks. We have quantified the overhead of our solution in terms of computation time and memory occupancy. The results of implementation of our proposal are obtained through real measurements on testbeds using TelosB motes.

Keywords: wireless sensor network, security of communications, public key management, key establishment, ECC, ECDH, ECAES, AES-CTR.