N° d'ordre :

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE D'ENSEIGNEMENT SUPERIEURE ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE DES SCIENCES ET DE LA TECHNOLOGIE
HOUARI BOUMEDIEN
FACULTE D'ELECTRONIQUE ET D'INFORMATIQUE

MEMOIRE

Présenté pour l'obtention du diplôme de MAGISTER

EN : Informatique

Spécialité : Informatique Mobile

Par BAGAA Miloud

Sujet

La sécurité de l'agrégation dans les réseaux de capteurs sans fil

Soutenu le 12/02/2008, devant le Jury composé de :

Mr Nadjib	BADACHE	Président
Mr Yacine	CHALLAL	Directeur de mémoire
Mr Omar	NOUALI	Examinateur
Mr Djamel	DJENNOURI	Examinateur

January 5, 2003

Je dédie ce travail à tous ceux qui me sont chers...

Ma mère;
Mon père;
Mes frères et soeurs;
Tous mes amis.

Miloud.

Remerciements

Je tiens à remercier mon Dieu, le tout puissant, de m'avoir donné le courage et la patience jusqu'à l'achévement de ce travail.

J'exprime ma profonde reconnaissance et mes vifs remerciements à mon directeur de thèse Dr Yacine CHALLAL, de m'avoir fait confiance en me proposant ce sujet. Je le remercie également pour ses lectures attentives et pour ses critiques et suggestions qui ont été d'un grand apport pour la finalité de ce travail. Je remercie Pr N. BADACHE, Dr D. DJENNOURI et Dr O. Nouali d'avoir accepté de juger ce travail.

Je voudrais également remercie mes parents, mes soeurs et mes frères, pour d'une part m'avoir patiemment supporté au cours de ces années.

J'adresse également mes sincères remerciements au Mr Abdelaziz ABDELOUAHAB, Directeur du la direction informatique du SONATRACH, pour sa compréhension, et pour tous les moyens qu'elle a mis à ma disposition.

Un grand MERCI, aux Noureddine LASLA, Mourad DELHOUM, AbdElraouf Ouadjaout, Amel BOUNOUA et Cherifa BOUDJADJA pour leurs encouragements, leur patience et leur amour.

Résumé

Un réseau de capteurs sans fil (RCSF) est un ensemble de capteurs communicants à travers des liaisons sans fils. Un capteur est une unité de calcul équipé d'un ensemble de dispositifs de captage (température, humidité, luminosité, vibration etc.), d'un moyen de stockage, et d'une batterie dont la durée de vie est limitée. Un tel réseau ne peut survivre si la perte de nœuds est trop importante car ceci engendre des pertes de communications dues à une trop grande distance entre les capteurs.

Pour cela les réseaux de capteurs sont déployés de manière dense pour minimiser le problème de déconnexion. Malheureusement, lorsque ce réseau devient dense, il consomme beaucoup d'énergie dans les communications. Les études montrent que plus de 70 % de l'énergie est consommée dans les transmissions des données. De plus les données sont redondantes, ainsi le nombre de collusions devient important, puisque le nombre de messages envoyés est énorme. Pour atténuer l'impact de ces problèmes sur la durée de vie du réseau l'agrégation de données est considérée comme une solution efficace. Au lieu que chaque nœud envoie ses données vers la station de base puis, cette dernière agrège les données, les données sont agrégées dans le réseau en utilisant des fonctions mathématiques comme **MIN**, **MAX**.... a la fin de ce processus la station de base reçoit une seule valeur qui représente une vue globale du phénomène capté par le réseau. L'agrégation de données permet de minimiser les collisions, aussi elle élimine la redondance des données brutes, et conserve l'énergie pour une plus longue durée de vie d'un réseau de capteurs [9].

L'agrégation nécessite que les données captées puissent être manipulées par les noeuds intermédiaires. D'où la difficulté de sécuriser les données agrégées par rapport aux mécanismes cryptographiques classiques, qui protègent les données de bout en bout.

Dans ce mémoire, nous avons étudié les différents protocoles de sécurité de l'agrégation de données, puis nous avons proposé un nouveau protocole appelé **SEDAN** (Secure and Efficient protocol for Data Aggregation in Wireless sensors Networks) [33], qui assure la sécurité de l'agrégation des données dans les **RCSF**. Notre protocole est basé sur le mécanisme de vérification

de l'intégrité des données à deux sauts. Notre solution est différente essentiellement par rapport aux solutions existantes, du fait qu'elle n'exige pas de communiquer avec la station de base pour vérifier et détecter les mauvaises agrégations. Ainsi, notre solution est basée sur un schéma totalement distribué pour garantir l'intégrité des données.

Nous avons comparé notre solution avec les autres protocoles que nous avons implémentés en utilisant l'environnement TinyOS. Les résultats de simulation montrent que le protocole proposé permet de faire des économies significatives dans la consommation d'énergie en préservant l'intégrité des données.

Publication

Nous avons publié notre protocole dans les actes de IEEE-LCN (Local Computer Networks) sous la référence :

M.Bagaa, N. Lasla, A. Ouadjaout and Y.Challal; "SEDAN : Secure and Efficient protocol for Data Aggregation in wireless sensor Networks", 32nd IEEE Conference on Local Computer Networks (LCN 2007) pp. 1053-1060, Worksohop on Netwok Security, October 2007.

Table des matières

Table des figures

Introduction Générale

Depuis quelques années, le marché des réseaux et des applications sans fil s'est considérablement développé. En particulier, une nouvelle branche s'est créée pour offrir des solutions économiquement intéressantes pour la surveillance à distance et le traitement des données dans les environnements complexes et distribués : les réseaux des capteurs sans fil.

Les capteurs offrent de façon usuelle un service (ex. les données qu'ils collectent) et utilisent les services fournis par d'autres nœuds pour construire des services plus complexes (ex. des services contrôlant des actuateurs en fonction des valeurs des capteurs). Le but des réseaux de capteurs est de bénéficier des effets de synergie d'un nombre gigantesque d'éléments dans un réseau dense.

L'agrégation de données intervient dans le transport des informations captées d'une manière généralement hiérarchique. Pour des raisons d'optimisation, ces données captées subissent des traitements intermédiaires afin de remonter l'information utile aux centres de prise de décision. Le paradigme de l'agrégation remplace les lectures des noeuds par une vue collaborative sur une zone spécifique. Cette collaboration implique que plusieurs noeuds participent, afin de calculer le résultat final de l'agrégation, qui permet aux noeuds agrégateurs de manipuler n'importe quelle valeur reçue dans sa région.

Un grand sous-ensemble d'applications utilisant les réseaux de capteurs nécessite de la sécurité, en particulier lorsque ces réseaux de capteurs protègent ou servent à la protection d'infrastructures critiques. La sécurité dans les réseaux de capteurs sans fils (RCSF) a six défis : (i) nature sans fil de communication, (ii) limitation de ressource des nœuds capteurs, (iii) nombre élevé de capteurs dans le réseau, (iv) manque d'infrastructure fixe, (v) topologie de réseau inconnue avant le déploiement, (vi) gros risque d'attaques physiques sur les capteurs sans surveillance

L'authentification et l'intégrité des échanges sont des services de sécurité indispensables pour certaines applications des réseaux de capteurs, notamment lorsqu'il s'agit de transporter des informations qui peuvent divulguer le secret médical, ou des informations sensibles quand il s'agit de prévenir des accidents catastrophiques comme dans les réacteurs nucléaire, etc... En cours de leur transport, des intrus tenteraient d'altérer le contenu de ces données ainsi que les résultats intermédiaires dans l'objectif d'inhiber ou fausser la prise de décision correcte. Par conséquent, il est très important de vérifier le comportement des noeuds agrégateurs et de détecter les noeuds qui veulent falsifier la vue collaborative. Afin de palier ce type d'attaque, il est nécessaire de sécuriser la propagation et l'agrégation des informations captées en assurant leur intégrité et leur authenticité.

Plusieurs contraintes rendent la sécurité de l'agrégation dans les RCSF, un challenge difficile à surmonter :

1) Les techniques traditionnelles de sécurité telles que la cryptographie à clé publique, exigent des calculs importants et l'envoi de longs messages qui peuvent épuiser rapidement les batteries des nœuds capteurs. Pour cela les protocoles de sécurité d'agrégation utilisent des mécanismes de sécurité légers (souvent symétriques).

2) Il est difficile de faire cohabiter et réconcilier la sécurité et l'agrégation.

Pour cela, dans le but de sécuriser l'agrégation, des protocoles de sécurité ont été proposés tel que **SAWN** [20], SRDA [21], **SDA** [24], **SDAP** [28] et **SecureDAV** [25]. Ces solutions peuvent être divisées en deux catégories sécurité de l'agrégation de bout en bout, et sécurité de l'agrégation de proche en proche. Dans la catégorie de protocoles qui sécurisent l'agrégation de proche en proche, la vérification de l'intégrité se fait dans le réseau et aussi dans la station de base. Par contre, dans les protocoles qui assurent la sécurité de l'agrégation de bout-en-bout, seule la station de base est responsable de la vérification de l'intégrité des données agrégées [6].

Ces solutions souffrent d'inconvénients majeurs qui sont :

i) la centralisation totale ou partielle de la vérification au niveau de la station de base, ce qui engendre des goulets d'étranglement et une mauvaise adaptation au facteur d'échelle.

ii) le rejet aveugle de données captées à cause d'une détection tardive de valeurs biaisées, ce qui entraîne le rejet de toute la masse de données "polluée".

Dans ce mémoire, nous avons conçu un nouveau protocole de sécurité de l'agrégation appelé **SEDAN** (Secure and Efficient Data Aggregation protocol for wireless sensor Networks) [33]. **SEDAN** est basé sur la vérification de l'intégrité à deux sauts ce qui évite de communiquer avec la station de base pour réaliser ces vérifications. **SEDAN**, prend en charge les problèmes décrits précédemment rencontrés dans **SAWN**, en utilisant ce nouveau type de clefs. Dans **SEDAN**, chaque nœud peut vérifier immédiatement les données envoyées par ses voisins de deux sauts, et les valeurs d'agrégation envoyées par ses voisins d'un saut. Cette amélioration élimine la transmission de fausses données, et ainsi optimise la consommation de l'énergie.

Organisation du document

Ce document est organisé comme suit :

Dans le chapitre 1, nous présenterons les caractéristiques et architectures des réseaux de capteurs.

Ensuite, nous aborderons les mécanismes de propagation et les protocoles d'agrégation de données dans les RCSF dans le chapitre 2.

Dans le chapitre 3, nous étudierons quelques protocoles récents de sécurité d'agrégation des données dans RCSF.

Puis nous présenterons notre protocole **SEDAN** dans le chapitre 4.

Enfin, nous terminons ce mémoire avec des résultats de simulations qui nous permettrons de comparer notre protocole avec d'autres protocoles de la littérature.

1 Les réseaux de Capteurs

1.1 Introduction

Les récentes avancées dans les domaines des technologies sans-fil et électroniques ont permis le développement à faible coût de minuscules capteurs consommant peu d'énergie (solution *low-cost* et *low-power*). Ces capteurs ont trois (3) fonctions :

- Capter des données (de type son, vibration, lumière,$\cdots$).
- Calculer des informations à l'aide de ces valeurs collectées.
- Les communiquer à travers un réseau de capteurs.

Les **RCSF** (Réseaux des Capteurs Sans Fil) utilisent un grand nombre de dispositifs très petits, nommés (nœuds de capteurs) , pour former un réseau sans infrastructure établie. Dans ces réseaux, chaque nœud est capable de détecter son environnement et de traiter l'information au niveau local ou de l'envoyer à un ou plusieurs points de collecte, à l'aide d'une connexion sans fil. Afin de résister aux chocs éventuels lors du déploiement, ces capteurs doivent être très solides et de plus, ils doivent aussi pouvoir survivre dans les conditions les plus extrêmes dictées par leur environnement d'utilisation (feu ou eau par exemple). En plus des contraintes environnementales, une contrainte très importante est l'économie d'énergie. En effet, un réseau de capteurs ne peut survivre si la perte de nœuds est trop importante car ceci engendre des pertes de communication dues à une trop grande distance entre les capteurs. Donc il est très important que les batteries durent le plus longtemps possible étant donné que dans la plupart des applications ils

sont placés aléatoirement (il serait alors impossible de retourner changer les batteries). Cette utilisation liée à l'autonomie des capteurs (1 année maximum pour les technologies actuelles) fait intervenir un paramètre non négligeable qui est le prix. Aucune application ne serait rentable si le rapport heures d'utilisation / prix était trop élevé. Il a donc été nécessaire d'allier technologie et low-cost. Puisque les réseaux de capteurs sont capables de surveiller leur environnement et de communiquer des données détaillées, ils peuvent comporter de nombreux avantages pour une grande diversité de secteurs. Ils peuvent avoir plusieurs domaines d'application, tels que [1] :

- Découvertes de catastrophes naturelles.
- Détection d'intrusions.
- Contrôle de la pollution.
- Agriculture.
- Applications de santé.
- Contrôle d'édifices.

1.2 Architecture

Les nœuds capteurs sont organisés en réseaux de capteurs (voir figure 1.1). Chacun de ces réseaux a la capacité de collecter des données et de les transférer au nœud de synchronisation central (**PUITS**) par l'intermédiaire d'une architecture multisaut. Le **PUITS** transmet ensuite ces données par Internet ou par satellite à l'ordinateur central (**gestionaire des taches**).

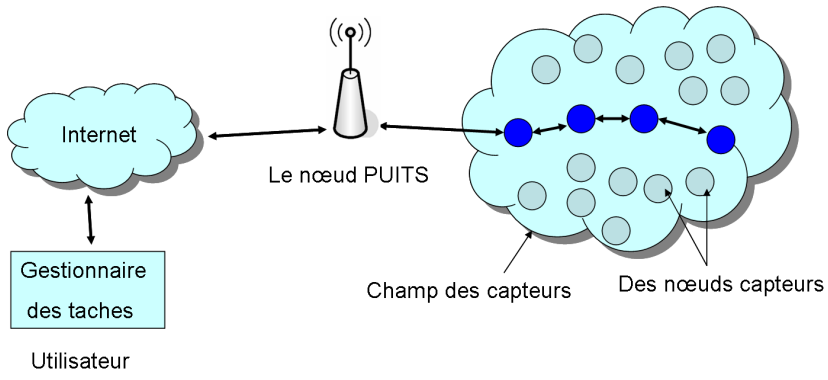


FIGURE 1.1: Architecture d'un réseau de capteurs [2]

Selon [5] il existe deux types d'architectures pour les réseaux de capteurs sans fil :

1. **Les réseaux de capteurs sans fil plats :**
   Un réseau de capteurs sans fil plat est un réseau homogène, où tous les nœuds sont

identiques en terme de batterie et de complexité du matériel, excepté le **PUITS** qui joue le rôle d'une passerelle et qui est responsable de la transmission de l'information collectée à l'utilisateur final. Selon le service et le type de capteurs, une densité de capteurs élevée (plusieurs nœuds capteurs/m^2) ainsi qu'une communication multisaut peut être nécessaire pour l'architecture plate. En présence d'un très grand nombre de nœuds capteurs, la scalabilité devient critique. Le routage et le contrôle d'accès au médium (**MAC**) doivent gérer et organiser les nœuds d'une manière très efficace en terme d'énergie.

2. **Les réseaux de capteurs hiérarchiques :**
   Une architecture hiérarchique était proposée pour réduire le coût et la complexité de la plus part des nœuds capteurs en introduisant un ensemble de nœuds capteurs plus coûteux et plus puissants, ceci en créant une infrastructure qui décharge la majorité des nœuds simples a faible coût de plusieurs fonctions du réseau. L'architecture hiérarchique est composée de multiples couches : une couche de capteur, une couche de transmission et une couche de point d'accès. Cette architecture sans-fil est influencée par un certain nombre de facteurs et contraintes tels que la tolérance de fautes, le redimensionnement, les coûts de production, l'environnement, la topologie de réseau, les contraintes matérielles, les médias de transmission et la consommation d'énergie.

Les principaux facteurs et contraintes influençant l'architecture des réseaux de capteurs sont détaillés ci-dessous :

- **La tolérance de fautes** [2]
  Certain nœuds peuvent générer des erreurs ou ne plus fonctionner à cause d'un manque d'énergie, un problème physique ou une interférence. Ces problèmes ne doivent pas affecter le reste du réseau, c'est le principe de la tolérance de fautes. La tolérance de fautes est la capacité de maintenir les fonctionnalités du réseau sans interruptions dues à une erreur intervenue sur un ou plusieurs capteurs.
- **L'échelle** [2]
  Le nombre de nœuds déployés pour un projet peut atteindre le million. Un nombre aussi important de nœuds engendre beaucoup de transmissions inter nodales (implémentation d'une détection d'erreur, d'un contrôle de flux,$\cdots$) et nécessite que le **PUITS** soit équipé de beaucoup de mémoire pour stocker les informations reçues.
- **Les coûts de production** [2]
  Souvent, les réseaux de capteurs sont composés d'un très grand nombre de nœuds. Le prix d'un nœud est critique afin de pouvoir concurrencer un réseau de surveillance traditionnel. Actuellement un nœud ne coûte souvent pas beaucoup plus que 1$. A titre de comparaison, un nœud bluetooth, pourtant déjà connu pour être un système

low-cost, revient environ à 10$.

- **L'environnement** [2]

  Les capteurs sont souvent déployés en masse dans des endroits tels que des champs de bataille au delà des lignes ennemies, à l'intérieur de grandes machines, au fond d'un océan, dans des champs biologiquement ou chimiquement souillés,$\cdots$ Par conséquent, ils doivent pouvoir fonctionner sans surveillance dans des régions géographiques éloignées.

- **La topologie de réseau** [2]

  Le déploiement d'un grand nombre de nœuds nécessite une maintenance de la topologie. Cette maintenance consiste en trois phases :

  - Déploiement.
  - Post-déploiement (les capteurs peuvent bouger, ne plus fonctionner,$\cdots$).
  - Redéploiement de nœuds additionnels.

- **Les contraintes matérielles** [2]

  La principale contrainte matérielle est la taille du capteur ($1cm^3$). Les autres contraintes sont que la consommation d'énergie doit être moindre pour que le réseau survive le plus longtemps possible, qu'ils s'adaptent aux différents environnements (fortes chaleurs, eau,$\cdots$), qu'ils soient autonomes et très résistants vu qu'ils sont souvent déployés par avion.

- **Les médias de transmission** [2]

  Dans un réseau de capteurs, les nœuds sont reliés par une architecture sans-fil. Pour permettre des opérations sur ces réseaux dans le monde entier, le média de transmission doit être normé. On utilise le plus souvent l'infrarouge (qui est license-free, robuste aux interférences, et peu onéreux), le bluetooth et les communications radio.

- **La consommation d'énergie** [2]

  Un capteur, de par sa taille, est limité en énergie (< 1.2V). Dans la plupart des cas le remplacement de la batterie est impossible. Ce qui veut dire que la durée de vie d'un capteur dépend grandement de la durée de vie de la batterie. Dans un réseau de capteurs (multisaut) chaque nœud collecte des données et envoie/transmet des valeurs. Puisque la puissance d'une transmission radio sans fil est proportionnelle au carrée de la distance ou d'ordre plus grand en présence d'obstacle, le routage par saut multiple consommera moins d'énergie qu'une communication directe. En fin toutes ces opérations sont gourmandes en énergie, c'est pour cette raison que les recherches actuelles se concentrent principalement sur les moyens de réduire cette consommation.

- **Le déploiement des nœuds** [7]

Le déploiement peut être soit déterministe ou auto-organisateur. Dans le premier cas, les capteurs sont placés manuellement et les chemins de routage sont prédéfinis. Dans l'autre cas, les nœuds capteurs sont éparpillés aléatoirement formant une infrastructure de manière ad hoc.

- **Modèle de livraison des données** [7]
  Selon l'application, le modèle de livraison des données au **PUITS** peut être continu, commandé par évènement, commandé par requête ou hybride.
- **Les capacités des nœuds** [6]
  Un nœud capteur peut être dédié à une fonction particulière tel que la retransmission, le captage et l'agrégation puisque l'affectation des trois fonctions à un même nœud peut épuiser rapidement son énergie.
- **agrégation et fusion des données** [6]
  Les nœuds capteurs peuvent générer des données redondantes, les paquets similaires provenant des différents nœuds peuvent être agrégés pour réduire le nombre de transmissions. Sachant qu'un calcul consomme moins d'énergie qu'une communication, des économies d'énergie considérables peuvent être obtenues par l'agrégation des données. On parle de fusion de données quand un nœud est capable de produire un signal plus précis en diminuant le bruit et en utilisant quelque technique pour combiner des signaux.
- **Hétérogénéité des nœuds** [6]
  Selon l'application un nœud capteur peut avoir différents rôles. L'existence de l'ensemble hétérogène de capteurs soulève beaucoup de questions techniques liées au routage des données.
- **Support de transmission** [6]
  Dans un réseau de capteur multisaut, les nœuds communicants sont liés par un médium radio. Les problèmes classiques associés aux supports sans fil (fading, taux d'erreur élevé $\cdots$) peuvent également affecter le fonctionnement du réseau de capteur.
- **Connectivité** [6]
  La densité élevée des nœuds dans les réseaux de capteur les empêchent d'être complètement isolés les uns des autres. Par conséquent, il est nécessaire que les nœuds de capteur soient fortement connectés.
- **couverture** [6]
  Dans les réseaux de capteur, chaque nœud obtient une certaine vue de l'environnement. La couverture de l'espace est également un paramètre de conception important dans le routage.
- **Qualité de service** [6]

Dans certaines applications, les données devraient être fournies au cours d'une certaine période du moment où elles sont captées, autrement les données seront inutiles.

1.3 Le Modèle

Ian et al [2] ont proposé un modèle de communication pour les RCSF. Le rôle de ce modèle consiste à standardiser la communication entre les participants afin que différents constructeurs puissent mettre au point des produits (logiciels ou matériels) compatibles. Ce modèle combine puissance et routage, intègre un protocole réseau, utilise les technologies sans-fil et favorise la coopération des nœuds de capteurs. Ce modèle comprend 5 couches qui ont les mêmes fonctions que celles du modèle **OSI** ainsi que 3 couches pour la gestion de la puissance, la gestion de la mobilité ainsi que la gestion des tâches. Le but d'un système en couches est de séparer le problème en différentes parties (les couches) selon leur niveau d'abstraction. Chaque couche du modèle communique avec une couche adjacente (celle du dessus ou celle du dessous). Chaque couche utilise ainsi les services des couches inférieures et en fournit à celle de niveau supérieur. Voici les 5 couches identiques à celles du modèle **OSI** plus en détails :

1. **La couche physique**
   Spécifications du câblage, des fréquences porteuses, etc$\cdots$
2. **La couche liaison**
   Spécifie comment les données sont expédiées entre deux nœuds/routeurs dans une distance d'un saut. Elle est responsable du multiplexage des données, du contrôle d'erreurs, de l'accès sur le media,$\cdots$ Elle assure la liaison point à point et multipoint dans un réseau de communication.

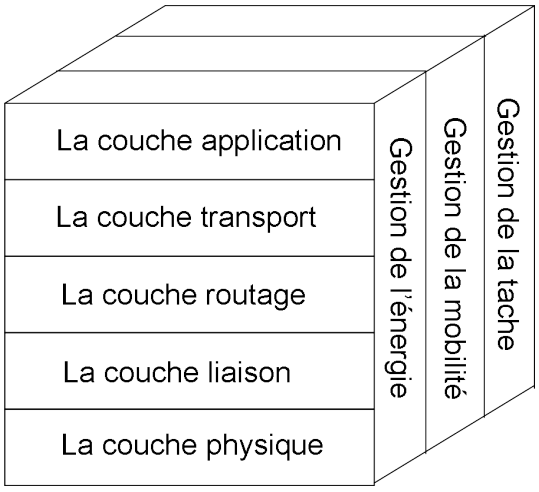


FIGURE 1.2: Modèle en couches d'une architecture de réseau de capteurs [2]

3. **La couche réseau**
   Dans la couche réseau le but principal est de trouver une route et une transmission fiable des données captées par les capteurs vers le **PUITS**. Ce routage diffère de celui des réseaux de transmission ad hoc sans fil typiques par les caractéristiques suivantes [7] :
   - il n'est pas possible d'établir un système d'adressage global pour le grand nombre de nœuds.
   - les applications des réseaux de capteurs exigent l'écoulement des données mesurées de sources multiples à un **PUITS** particulier.
   - les multiples capteurs peuvent produire des mêmes données à proximité d'un phénomène (redondance).
   - les nœuds capteur exigent ainsi une gestion soigneuse des ressources.
4. **La couche transport**
   Cette couche est chargée du transport des données, de leur découpage en paquets, du contrôle de flux, de la conservation de l'ordre des paquets et de la gestion des éventuelles erreurs de transmission.
5. **La couche application**
   Cette couche assure l'interface avec les applications. Il s'agit donc du niveau le plus proche des utilisateurs, géré directement par les logiciels.

1.4 TinyOS (système d'exploitation réduit)

TinyOS [3] est un système d'exploitation intégré, modulaire, destiné aux réseaux de capteurs miniatures. Cette plate-forme logicielle ouverte est une série d'outils développés par l'Université de Californie à Berkeley et enrichie par une multitude d'utilisateurs. En effet, TinyOS est le plus répandu des OS pour les réseaux de capteurs sans-fil. Il est utilisé dans les plus grands projets de recherches sur le sujet (plus de 10,000 téléchargements de la première version). Un grand nombre de ces groupes de recherches ou entreprises participent activement au développement de cet OS en fournissant de nouveaux modules, de nouvelles applications,$\cdots$ Cet OS est capable d'intégrer très rapidement les innovations en relation avec l'avancement des applications et des réseaux eux même tout en minimisant la taille du code source en raison des problèmes inhérents de mémoire dans les réseaux de capteurs. La librairie TinyOS comprend les protocoles réseaux, les services de distribution, les drivers pour capteurs et les outils d'acquisition de données. TinyOS est en grande partie écrit en C mais on peut très facilement créer des applications personnalisées en langages *C*, *NesC*, *Java*.

1.5 Conclusion

La flexibilité, la tolérance de fautes, le prix réduit et les caractéristiques rapides de déploiement des réseaux de capteurs offrent des possibilités énormes de développement dans tous les domaines d'application. Cependant, la réalisation des réseaux de capteurs doit satisfaire quelques contraintes parmi lesquelles on peut citer la consommation d'énergie, le prix du matériel, le changement de topologie et l'adaptation à l'environnement. Ces contraintes exigent que des nouvelles techniques de gestion de réseau sans-fil soient mises au point. Ceci explique pourquoi cette nouvelle technologie est malheureusement encore trop peu répandue sur le terrain mais est cantonnée principalement dans les laboratoires de recherche. Néanmoins, les avancées technologiques permettent de penser que les réseaux de capteurs feront bientôt partie intégrante de nos vies et satisferont sûrement les plus grands projets.

2

Les mécanismes d'agrégation et de propagation

2.1 Introduction

Dans les réseaux de capteurs, le coût de communication est souvent plus grand que le coût de calcul. Pottie et Kaiser [4] affirment que la consommation d'énergie pour l'exécution de 3 millions instructions est équivalente à l'envoi de *1 Koctets* de données sur une distance de *100 mètres*. Pour optimiser le coût de communication dans le réseau de capteurs, l'agrégation des données est considérée comme une technique efficace. Elle permet de minimiser les collisions. De plus, elle permet d'éliminer la redondance des données brutes. Telle opération est aussi utile pour l'extraction de l'information pour une application spécifique à partir des données brutes, ainsi que pour conserver l'énergie pour une plus longue durée de vie, d'un réseau de capteurs [9]. Par exemple, supposant que l'opérateur veut lire la moyenne d'une certaine valeur dans le réseau. Premièrement, une façon inefficace de trouver la moyenne, est que chaque nœud capteur envoie sa lecture à la station de base (probablement en utilisant les mêmes routes) puis la station de base effectue le calcule de la moyenne de toutes les lectures reçues. La deuxième façon qui est plus efficace, consiste à rassembler les mêmes informations, au niveau de noeuds intermédiaires qui calculerons la moyenne de ces valeurs et envoient seulement les valeurs moyennes et le nombre de messages reçus. Récemment plusieurs travaux ont aboutit à différents protocoles d'agrégation pour les réseaux de capteurs tout en supposant un environnement saint : **la diffusion**

dirigée [10], **LEACH** [11], **l'agrégation avide** [12] et **Couguar** [13].

2.2 Traitement des requêtes

Dans les anciennes approches, les nœuds capteurs avaient un programme incorporé pour traiter les données et les envoyer au **PUITS** (station de base) via les protocoles du réseau. L'inconvénient de cette approche est que l'utilisateur ne peut pas changer le comportement du système. Cela exige une interface qui sépare le traitement des requêtes et les protocoles du réseau.

2.2.1 Modèles de Requête

COUGAR [13] est une solution qui prévoit une couche de requêtes pour traiter les requêtes d'agrégation. Avec l'interface offerte, les clients peuvent effectuer des requêtes sans savoir comment les résultats seront produits, traités et retournés par le réseau. La couche requête traite des requêtes déclaratives et produit un plan de requêtes de coût optimal, avec une approche similaire aux bases de données. Cependant, la vue du coût est différente pour les réseaux de capteurs. Le facteur principal à prendre en considération est le coût de communication, impliquant le coût du routage et de l'agrégation des données. Un plan de requête décide la quantité de calcul à appliquer sur le réseau et spécifie le rôle et la responsabilité de chaque nœud capteur.

2.2.2 Les requêtes et l'agrégation

On peut classifier les requêtes pour les réseaux de capteurs en trois catégories : [7]

- **Requêtes simples :**
  Ceux sont des requêtes qui n'utilisent pas l'agrégation, par exemple :

  SELECT température
  FROM capteur
  WHERE nœud = z.

  Elles sont généralement utilisées, soit à la diffusion ou point à point pour diriger la requête.
- **Requêtes complexes :**
  Elles peuvent contenir des requêtes secondaires, par exemple :

SELECT température
FROM capteur
WHERE chambre **IN** (**SELECT** chambre
FROM immeuble
WHERE étage = 3
).

– **Requêtes entraînées par les événements :**
Ce sont des requêtes continues qui renvoient des valeurs périodiquement à un intervalle bien spécifique, Par exemple :

SELECT région, **AVG**(son)
FROM capteur
GROUP BY région
HAVING AVG (son) > 200
SAMPLE PERIOD 10.

L'interface de requête soutient un langage semblable à celui d'**SQL** pour exprimer des requêtes d'agrégation, contenant des clauses **SQL** comme **SELECT**, **GROUPE BY**, **HAVING** et les clauses d'agrégation comme **MAX**, **AVG**, **MIN**, **COUNT** et **SUM**. On ajoute pour les requêtes de surveillance continue, des clauses comme **DURATION** et **EVERY** [15]. Pendant cette durée, le nœud peut se mettre en veille, ceci peut conserver l'énergie d'une durée plus longue. Les clauses comme **SUM**, **AVG**, **COUNT** sont sensibles aux duplications de données, par contre les clauses **MIN**, **MAX** ne le sont pas.

2.3 Les différentes approches d'agrégation de données

On peut classer les différentes techniques d'agrégation de données dans les réseaux de capteurs en deux approches, comme illustre la Figure 2.1.

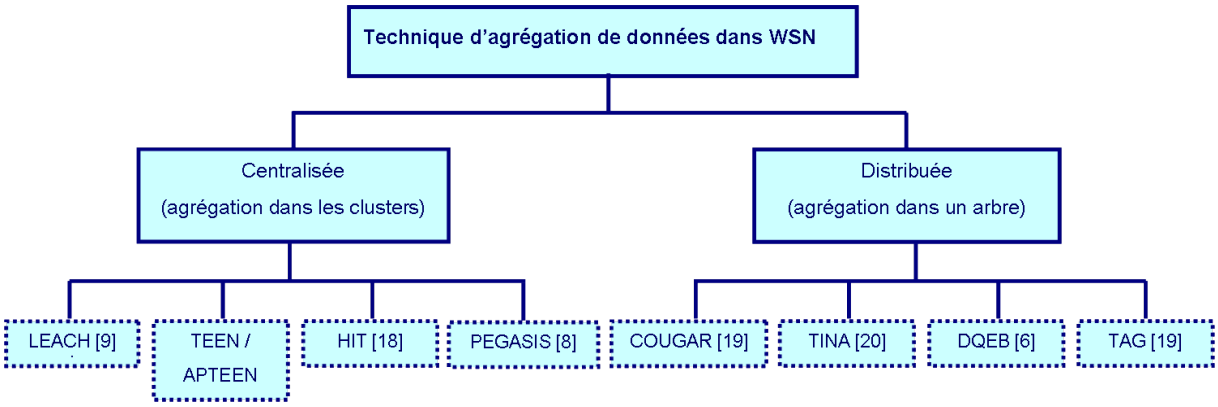


FIGURE 2.1: Classification de différentes techniques d'agrégation dans les RCSF

2.3.1 Approche centralisée

C'est une approche où chaque nœud envoie des données à un nœud central en utilisant la route optimale en utilisant des protocoles de routage multisaut. Les nœuds capteurs envoient simplement les paquets de données à un leader qui est un nœud puissant. Le leader agrège les données qui peuvent être demandées.

Chaque nœud envoi des paquets de données à travers des nœuds intermédiaires au nœud leader, ainsi un grand nombre de messages doivent être transmis pour une requête. Dans les meilleurs cas, il est égal à la somme des longueurs de chemin pour chaque nœud. L'inconvénient majeur de cette approche est que les réseaux de capteurs exigent à prendre en considération la contrainte de l'énergie et par conséquent l'approche est coûteuse et engendre l'échange de beaucoup de message pour chaque requête. Une amélioration de cette approche est représentée par les protocoles hiérarchiques. **LEACH** [9] et **PEGASIS** [8] appartiennent à cette famille.

1. **le protocole LEACH :**
   Dans [9], les auteurs proposent le protocole **LEACH** qui se base sur la construction de clusters, et laisse l'agrégation des données aux chefs de clusters qui communiquent directement avec la station de base. Pour distribuer la consommation de l'énergie de manière équitable sur tous les nœuds, le chef de cluster est aléatoirement élu dans chaque cluster. Dans [11], les auteurs proposent une version modifiée nommée **LEACH-C**. Cette dernière utilise la station de base pour diffuser la nomination du chef de cluster, ce qui rend la consommation d'énergie équitable.

   Basé sur **LEACH**, [16] raffine l'algorithme d'élection du chef de cluster en laissant chaque nœud diffuser le message *HELLO* et compter ses voisins à chaque étape d'initialisation, où les nœuds les plus qualifiés et potentiels (ayant plus d'énergie) deviennent les chefs de clusters. Cette modification disperse les chefs de clusters de manière équitable dans le réseau sans exiger la participation de la station de base, de plus, chaque nœud diffuse le message *HELLO* par sa puissance de transmission maximale dans l'étape d'initialisation de chaque cluster. Il réalise seulement une amélioration légère (presque 6 % en terme de consomation d'énergie) sur **LEACH**.
2. **le protocole PEGASIS**
   La réduction du nombre de chef de cluster est critique pour conserver l'énergie puisque ces nœuds restent éveillés et aussi transmettent à la station de base avec une puissance élevée.

Lindsey et al, ont conçu **PEGASIS**[8], qui organise tous les nœuds dans une chaîne, chacun peut jouer le rôle de chef de cluster à tour de rôle. Puisqu'il y a seulement un chef de cluster dans PEGASIS et Ce dernier est le seule responsable de l'envoie des données à la station de base, il n'y a aucune transmission simultanée, ce qui élimine le temps de latence.

3. **le protocole HIT**
   Basé sur **LEACH** et **PEGASIS**, Culpepper et al, ont proposés la transmission indirecte hybride (**HIT**) [18]. **HIT** utilise toujours des clusters comme **LEACH**, mais permet des routes de multi saut entre les chefs de cluster et les capteurs. **HIT** améliore **LEACH** et **PEGASIS**, en permettant le routage multisauts entres les chefs de cluster, ce qui assure le passage à l'échelle.
4. **les protocoles TEEN / APTEEN**
   TEEN (Threshold sensitive Energy Efficient sensor Network protocol) est un protocole hiérarchique conçu pour être sensible aux changements soudains des attributs captés tels que la température. L'architecture du réseau est basée sur un groupement hiérarchique où les nœuds les plus proches forment des clusters. Après la construction des clusters, le chef de cluster diffuse deux seuils aux nœuds, qui sont la valeur minimale d'un attribut pour pouvoir être transmit et le degré minimal du changement de cet attribut. Le **TEEN** adaptatif (**APTEEN**) est une extension de **TEEN** basée sur le captage périodique des données et la réaction aux événements temps-réel. Quand la station de base forme les clusters, les chefs de cluster diffusent les attributs, les seuils et le plan de transmission à tous les nœuds et effectuent également l'agrégation des données afin d'économiser de l'énergie.

2.4 L'approche distribuée

C'est une approche centrée-données où les nœuds intermédiaires peuvent consulter le contenu et agréger les données transportées par les différents paquets qu'ils réceptionnent. La motivation principale de l'agrégation distribuée est que le coût de communication est plus important que le coût de calcul. Elle implique le décalage d'une partie du calcul des clients aux nœuds capteurs agrégeant les résultats ou filtrant les données inutiles, ceci dans le but de réduire le transfert des messages et l'utilisation efficace de la bande passante, pour conserver l'énergie pour une plus longue durée de vie d'un réseau de capteurs. Il peut y avoir deux variations pour l'agrégation dans le réseau :

- **Le fusionnement des paquets**
  L'envoi de différents paquets encourt le surcoût d'en-têtes de paquet chaque fois. Une meilleure approche serait d'agréger les paquets et d'envoyer un grand paquet

simple. Ceci réduira le coût lié aux en-têtes de paquets.

– **L'agrégation partielle**

Pour des requêtes d'agrégation, les nœuds intermédiaires peuvent calculer les résultats partiels qui peuvent être utilisés pour calculer les résultats finaux. Ceci économise considérablement l'énergie consommée. La limitation d'une telle agrégation est la gestion fondamentale du réseau, le protocole doit fournir le support pour la synchronisation. En outre la synchronisation augmente le temps de réponse des requêtes. L'arbre de recouvrement est considéré comme la meilleure structure de routage pour une agrégation optimale. Parmi les protocoles de cette approche on cite :

1. **Le protocole Cougar :**

   Cougar [13] a modélisé les données produites par le réseau de capteurs comme une table relationnelle. Dans cette table, chacun des attributs représente soit des informations sur le nœud capteur ou bien des données produites par ce nœud. L'approche **Cougar** [13] fournit une agrégation partielle au niveau des nœuds fils. Chaque nœud maintient une liste d'attente contenant les nœuds fils qui doivent lui envoyer les paquets. Le nœud n'émet le paquet agrégé au prochain saut que s'il a reçu les paquets de tous les nœuds de la liste d'attente. Cependant, un nœud peut devenir inaccessible à cause du mouvement ou d'un problème de batterie. Pour cela, il faut utiliser un *Timer* afin d'éviter une attente indéfinie. Après un certain délai, le nœud parent assume que le nœud fils est devenu inactif. Ceci exige la réparation des routes. **Cougar** propose d'utiliser la stratégie locale de réparation de route qui s'agit de trouver une nouvelle route dans son voisinage.

2. **Le protocole TAG**

   [14] propose l'approche TAG pour les réseaux de capteurs avec des nœuds hétérogène [1]. Le nœud racine initialise la diffusion par l'envoi d'un message avec le hop_count égale 0 et son sensor_Id. Pour chaque nœud capteur ayant reçu ce message, il incrémente le hop_count, attache son identité et le rediffuse encore. Il choisit la source du message comme son père. Le processus continue en bas de l'arbre.

   Le traitement d'une requête implique deux phases :

   – La propagation des requêtes.

   – L'agrégation du résultat.

Dans **TAG**, une modification du schéma d'agrégation peut être utilisée pour des requêtes de surveillances continues. La requête est lancée au niveau de la racine. En réceptionnant la requête par les nœuds fils immédiats, ils envoient leurs valeurs

1. Ils n'ont pas la même capacité de stockage et les mêmes caractéristiques en énergie.

courantes agrégées à la racine. Egalement, ils diffusent la requête aux nœuds fils. Ainsi après le premier saut, la racine obtient l'agrégat des fils immédiats. Pendant le prochain saut, les fils de nœud racine obtiendront les paquets de leurs fils et envoient les valeurs agrégées au nœud racine. Ainsi la racine obtiendra les valeurs agrégées.

Après N sauts, le nœud racine obtient l'agrégation du n^{eme} niveau. La première agrégation complète génère un grand nombre de messages. Mais ensuite, chaque nouvelle agrégation génère n messages où n c'est la profondeur de l'arbre.
La méthode *Snooping*[2] peut également être utilisée pour agréger les données. La requête d'un client est livrée à la racine. La racine envoie ses propres valeurs. Le paquet envoyé par la racine atteint également ses fils. Quand un fils S_i entend la racine envoyant ses données, il assume qu'il doit aussi envoyer des données et envoie à son tour ses valeurs à la racine. Les fils de S_i suivent la même chose et le processus continu. Ceci économise les messages utilisés pendant la phase de propagation. Pour les requêtes *Min* et *Max*, le *Snooping* est une bonne approche : Chaque nœud écoute ses voisins lorsqu'ils envoient les données. Ce noeud n'enverra sa valeur que si elle est meilleure (min ou max selon la fonction d'agrégat) que la valeur écoutée. Une autre approche est de trouver la valeur *Min* ou *Max* jusqu'au k^{eme} niveau et puis demander aux nœuds d'envoyer la valeur inférieure ou supérieure à la valeur observée. Pour des requêtes de groupe, chaque enregistrement d'agrégation partiel est assigné à une identification de groupe basée sur l'expression de groupe. Lorsque un nœud entend des paquets provenant de ses fils, si l'identificateur de groupe transmit dans le paquet correspond à son propre identificateur de groupe, il combine la valeur sinon il la stocke directement pour l'émission. Dans le cas d'une clause de requête, un filtre est appliqué aux agrégats du groupe créé à chaque nœud. Les groupes peuvent être grands et par conséquent la mémoire centrale peut faire défaut. Pour éviter cela, la pré agrégation est utilisée. TAG offre Beaucoup d'avantages tel que :

(a) Il économise l'énergie.

(b) Réduit au minimum le nombre de messages transférés.

(c) Utilise des délais qui permettent aux nœuds de dormir.

3. **Le protocole TINA**

TINA [19] fournit d'autres optimisations au-dessus de **COUGAR** [13] et **TAG** [19]. Il exploite les tolérances temporelles de concordance, qui réduit plus loin la consommation d'énergie de 60% et prolonge-la duré de vie de 300%. L'approche consiste à envoyer les données, seulement, à l'apparition d'un changement significatif au niveau des valeurs de données. Il utilise le concept de *Timer*. Il est également

2. Chaque nœud écoute ses voisins lorsqu'ils envoient des messages

utilisé ici pour synchroniser la réception des paquets provenant des nœuds fils et à l'envoi des données agrégées. Dans **TINA**, avec la clause **WHERE** une condition *tct* est donnée. Cette condition dit qu'une valeur de donnée peut être ignorée si la différence entre celle-ci et la précédente est inférieure ou égale à la valeur indiquée par *tct*. La clause **WHERE** filtre les données qui ne satisfont pas la condition et le *tct* filtre les données dont la valeur est dans le rang de tolérance indiquée. Ceci exige à chaque nœud, des conditions plus élevées de mémoire parce qu'il doit stocker les résultats intermédiaires des nœuds fils (agrégats partiels).En effet, premièrement le nœud compare la vue stockée avec les informations reçues. S'ils ne sont pas dans le rang spécifié par *tct*, les données sont envoyées. L'avantage est la réduction significative du nombre de messages par rapport à **COUGAR** et **TAG**.

4. **DQEB (Dynamic Query tree Energy Balancing Protocol)**
   La plupart des protocoles ci-dessus supposent que la construction des arbres est statique. Ils supposent que la perte d'énergie est la même au niveau des nœuds. En pratique, les nœuds initialement ont le même potentiel en énergie, mais avec le temps, et à force que ces derniers transmettent et réceptionnent des données, ils perdent cette énergie, et sur tous quand il s'agit des nœuds (*pères*).

   DQEB [6] propose une approche d'énergie équilibrée et modifie dynamiquement la structure de l'arbre. Cette dernière est basée sur le seuil d'énergie des nœuds. Les auteurs supposent que les nœuds sont organisés en clusters et chaque cluster possède un chef (*leader*). Chaque nœud possède un poids qui s'incrémente avec le temps. Ce poids présente la quantité de consommation de l'énergie.

   Quand l'énergie d'un nœud diminue, il est plus sage de le déplacer en bas, c'est-à-dire, il devien un nœud feuille, cela est en vue d'éviter la répartition de l'arbre car cette dernière est coûteuse.

   Puisque les nœuds possédant moins d'énergie deviennent des nœuds feuilles, ils vivront un peu plus, parce qu'ils envoient seulement leurs lectures, ce qui augmente la duré de vie du réseau.

2.5 Conclusion

Les protocoles d'agrégation de données étudiés dans ce chapitre, quelque soit leurs approche (centralisée, distribuée), supposent un environnement saint, où chaque nœud peut envoyer des données agrégées, ou bien des lectures, sans solliciter les services de

sécurité (authentification, confidentialité, intégrité $\cdots$ etc). Cependant, ces protocoles sont déployés dans les réseaux de capteurs, qui sont utilisés dans des domaines critiques, à l'exemple du domaine militaire. Dans un tel environnement, un adversaire peut placer plusieurs nœuds malicieux qui transmettent de faux messages dans le réseau. De plus, un tel adversaire peut accéder à la clé physique d'un nœud. Le nœud malicieux, peut toucher à l'intégrité des données. Par exemple, il peut modifier une lecture, comme il peut changer une valeur agrégée, ce qui est plus dangereux. Dans le prochain chapitre, nous traiterons la problématique de sécurité de l'agrégation de données en terme d'intégrité et d'authentification.

3

Description et analyse d'un ensemble de protocoles de sécurité d'agrégation des données

3.1 Introduction

L'agrégation de données est une collection des lectures de capteurs, elle représente une vue collaborative d'un ensemble de nœuds. Généralement, elle est appliquée sur une zone très spécifique limitée par le nœud **PUITS**. Ce dernier envoie la valeur de l'agrégation à la station de base en utilisant un protocole de routage. Le nombre de nœuds qui envoient leurs lectures peut être important et les informations envoyées par ces nœuds peuvent être redondantes. L'agrégation de données conserve l'énergie par la réduction du nombre de données acheminées par les nœuds intermédiaires.

L'agrégation nécessite que les données captées puissent être manipulées par les noeuds intermédiaires. D'où la difficulté de sécuriser les données agrégées par rapport aux mécanismes cryptographiques classiques, qui protègent les données de bout en bout.
Il est difficile de faire cohabiter et réconcilier la sécurité et l'agrégation :

- On ne peut pas prendre le risque de stocker la même clé dans chaque nœud pour permettre le chiffrement et l'authentification, du fait qu'un adversaire peut récupérer la clé d'un nœud, ainsi il devient capable de contrôler le réseau entier.

– On ne peut pas crypter des messages en utilisant une clé unique partagée entre chaque nœud et le nœud **PUITS**, du moment où chaque nœud intermédiaire a besoin d'avoir accès au contenu des messages reçus pour exécuter l'agrégation.

Lorsqu'un intrus obtient la clé d'un nœud légitime, il peut faire des attaques sur le réseau par la falsification soit des lectures des nœuds ou des valeurs d'agrégation. De cette façon, l'objectif de la sécurité de l'agrégation est la détection des opérations d'agrégation corrompues, et/ou l'injection de fausses données pendant le processus d'agrégation.

Les protocoles de sécurité de l'agrégation des données fournissent des mécanismes de sécurité légers pour détecter efficacement le mauvais comportement des nœuds. Ces protocoles peuvent être classés en deux catégories selon le type de cryptage des données utilisé (voir la Figure 3.1) :

– Des protocoles basés sur le cryptage des données de *bout-en-bout*.

– Des protocoles basés sur le cryptage des données de *proche-en-proche*.

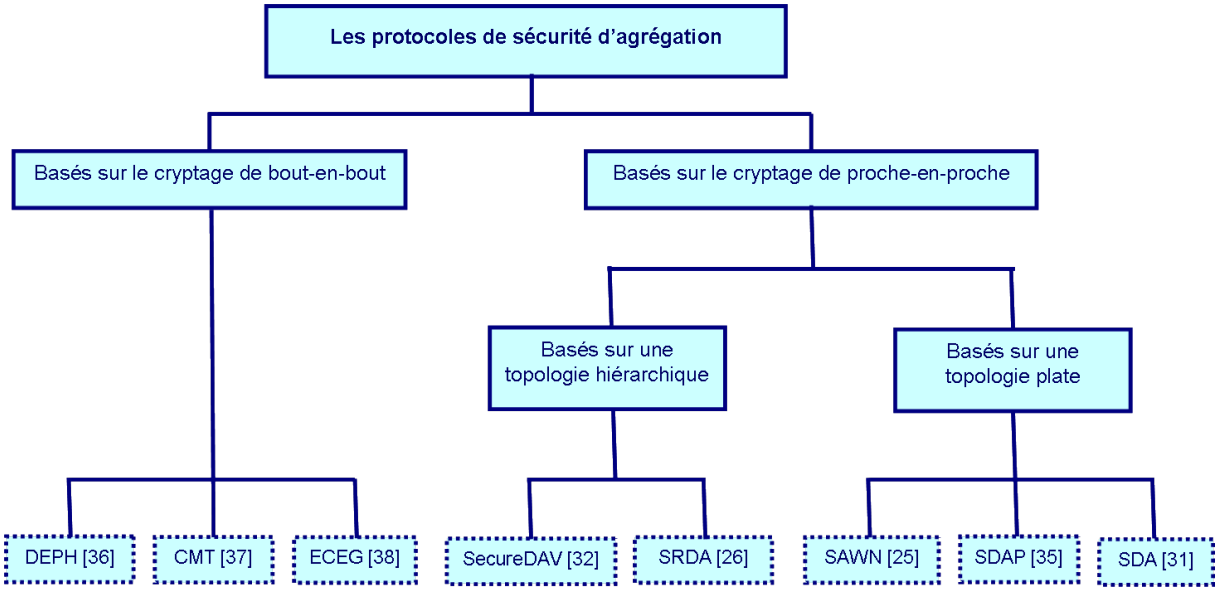


FIGURE 3.1: Classification des protocoles de sécurité d'agrégation dans les réseaux de capteurs

3.2 Menaces et objectifs de sécurité

L'objectif des protocoles de sécurité d'agrégation est de fournir des techniques d'agrégation qui résistent à le captage des noeuds. C'est à dire qu'un noeud compromis ne devrait pas avoir seul le pouvoir d'écouter des données, de falsifier des messages, ou d'empêcher les autres noeuds d'accéder aux données.

Même un unique agrégateur capté représente une menace sérieuse pour la sécurité d'un réseau de capteurs. En conséquence, il faut concevoir des techniques qui assurent

une sécurité raisonnable en présence d'agrégateur compromis. Idéalement, il faudrait que la sécurité du réseau se dégrade progressivement au fur et à mesure que des noeuds sont compromis. Dans ce mémoire, sécurité signifie résistance contre les attaques suivantes : écoute passive, falsification de données, injection de paquets, et déni de service.

Écoute passive : Une écoute passive se produit lorsqu'un attaquant capture un noeud et étudie le trafic qui le traverse sans en altérer le fonctionnement. Puisqu'un noeud d'agrégation traite des données venant de plusieurs noeuds dans le réseau, les informations qu'il envoie ne concernent pas uniquement ce noeud, mais plutôt un groupe de noeud.

Falsification de données et injection de paquets : Un nœud compromis peut modifier les paquets qui le traversent. Il peut aussi envoyer de faux messages. Puisqu'un message qui agrège des données embarque des informations qui concernent un ensemble de capteurs, il est plus intéressant pour un attaquant de falsifier de tels messages plutôt que la simple mesure d'un capteur. Un attaquant qui contrôle la signification des messages qu'il envoie peut avoir un impact lourd sur le résultat final calculé par le nœud **PUITS** (le puit).

Un attaquant qui ne contrôle pas la signification des messages falsifiés (par exemple, si le message est chiffré avec une clef inconnue de l'attaquant) peut tout de même faire des dégâts. Il peut envoyer du bruit sans signification (des déchets) et rendre le réseau inutilisable — cela peut être vu comme une forme de déni de service. Également, un type particulier d'injection de paquet est l'attaque par rejeu, au sein de laquelle un noeud mal-intentionné écoute des messages dans le but de les ré-envoyer plus tard.

Déni de service : Un noeud compromis peut arrêter d'agréger et d'acheminer les données. Ce faisant, il empêche le nœud **PUITS** de récupérer des informations au sujet de plusieurs noeuds dans le réseau. Si l'attaquant continue d'échanger des messages de routage en dépit de son comportement malhonnête, ce problème peut se révéler difficile à résoudre. De cette façon, un agrégateur mal-intentionné peut rendre le réseau inutilisable. Des attaques plus évoluées peuvent aussi ne jeter que certains messages d'une façon aléatoire. Il est également difficile de détecter quand un attaquant envoie des déchets. Finalement, il est important de réaliser que de telles attaques ne nécessitent pas forcément un coût élevé ou des compétences techniques avancées. Par exemple, une attaque primitive consisterait simplement à détruire le capteur physiquement.

3.3 Les protocoles basés sur le cryptage des données de bout-en-bout

Dans ce type de protocoles, le seul qui est capable de décrypter les données cryptées est le nœud **PUITS**. Les nœuds intermédiaires sont capables seulement de faire l'agrégation

sur ces données.

Les protocoles de cette catégorie utilisent une clé partagée entre chaque nœud et le nœud **PUITS** pour garantir l'intégrité des données transmises dans le réseau. Comme les contenus des données sont cryptés, les nœuds intermédiaires utilisent un type de cryptographie particulier appelé *Privacy Homomorphism (PH)* [32] pour pouvoir exécuter l'agrégation sans décrypter les données.

Un algorithme est *PH* si et seulement si en ayant $E(x)$ et $E(y)$ on peut calculer $E(x \oplus y)$ sans décrypter x et y. Ainsi il vérifie la propriété suivante :

$$E_{K_1}(x_1) \oplus E_{K_2}(x_2) = E_{K_1+K_2}(x_1 \oplus x_2).$$

Où K_i sont les clés et x_i sont les données. Le point unique de vérification dans ce type de protocoles est le nœud **PUITS**. Ce dernier ayant toutes les clés utilisées pour crypter les données dans le réseau.

Les avantages de cette approche sont :

- **Le traffic de réseau :** L'un des avantages principaux de ce type de protocoles est l'optimisation du traffic réseau ; au lieu que chaque nœud envoie ses données vers le nœud **PUITS** puis ce dernier fait l'agrégation, l'agrégation dans ce type de protocoles est faite dans le réseau. De plus, ces protocoles sont similaires à un processus d'agrégation simple.
- **L'effort de calcul :** En comparaison avec les protocoles basés sur le cryptage de *proche-en-proche*, ces protocoles font moins de calcul, puisque l'agrégation des données dans ces protocoles est faite sur les données cryptées sans besoin de décrypter et de crypter ces dernières dans les nœuds intermédiaires.
- **Sécurité :** un autre avantage de ces protocoles est la sécurité des données en comparaison avec les protocoles de cryptage de proche-en-proche. Ceci est dû au fait que les données ne sont pas décryptées au niveau de chaque nœud agrégateur où l'attaquant peut violer la confidentialité des données.

Il y a plusieurs protocoles proposés dans la littérature qui utilisent la méthode PH [29, 30, 31]. Cependant, quelques protocoles [31] utilisent des techniques de cryptographie à clé asymétrique comme *ElGamal* et d'autres [29, 30] utilisent des algorithmes de cryptographie symétrique. Ces protocoles utilisent un seul type de fonction d'agrégation qui est la somme.

3.3.1 Le protocole Domingo-Ferrer (DEPH)

Dans [29], l'auteur introduit un algorithme **PH** de schéma symétrique, l'idée de ce protocole est que chaque nœud P divise sa donnée m sur d parties et crypte chaque

partie en utilisant une clé symétrique partagée entre P et le nœud **PUITS**. Enfin, le noeud P envoie au nœud **PUITS** ces parties cryptées. Pendant la phase d'acheminement de données, l'agrégation est faite sur les parties de données cryptées. Lorsque le nœud **PUITS** reçoit les parties de la valeur d'agrégation, il vérifie leur intégrité et les regroupe pour trouver la valeur de l'agrégation finale.

L'algorithme suivant montre les différentes étapes de ce protocole.

L'algorithme de Domingo-Ferrer (2002) [29]

Paramètre :

La clé publique : un entier $d \geq 2$, un grand nombre entier M.

La clé secrète : g un nombre qui divise M ; un nombre r et son inverse r^{-1} existent dans Z_M.

Cryptage :

Diviser m sur d partie $m_1, ...m_d$ où $\sum_{i=1}^{d} m_i \ mod \ g \ = \ m$.

$C = [c_1, \ldots, c_d] = [m_1 \times r \ mod \ M, m_2 \times r^2 mod \ M, \ldots, m_d \times r^d mod \ M]$.

Décryptage :

$m = (c_1 \times r^{-1} + c_2 \times r^{-2} + \ldots \ + c_d \times r^{-d}) \ mod \ g$.

Agrégation :

L'addition modulo M.

$C_{12} = C_1 + C_2 = [(c_{11} + c_{21}) mod \ M, \ldots, (c_{1d} + c_{2d}) mod \ M]$.

Un inconvénient de ce protocole est la taille du message qui dépend de d. Si d est très grand alors la taille du message devient aussi grande et chaque nœud pour une seule fonction d'agrégation fait d fois l'agrégation. Ce protocole a aussi un autre inconvénient qui est l'attaque de rejeu, où un nœud malicieux peut sauvegarder un message m puis l'envoie dans un autre processus d'agrégation sans qu'il soit détecté par le nœud **PUITS**.

3.3.2 Le protocole CMT

Le protocole proposé dans [30] par Castelluccia, *Mykletun*, et *Tsudik*, est basé sur l'hypothèse que chaque nœud utilise une clé symétrique partagée entre ce nœud et le nœud **PUITS**, cette clé doit être chaque fois modifiée pour éviter l'attaque de rejeu. L'idée de ce protocole est que chaque nœud fait l'addition modulaire entre sa clé stockée et sa donnée. Pendant la phase d'acheminement des données, l'agrégation se fait sur ces données qui sont déjà cryptées.

L'algorithme suivant montre les différentes étapes de ce protocole

L'algorithme de CMT [30]

Paramètre :

Sélection d'un grand nombre entier M.

Cryptage :

Le message $m \in [0, M-1]$.

Aléatoirement générer une clé $k \in [0, M-1]$.

$C = (m + k) mod M$.

Décryptage :

$m = (c - k) mod M$.

Agrégation :

$c_{12} = (c_1 + c_2) mod M$.

La taille du paquet dans ce protocole dépend de la taille de M, et une seule addition modulaire suffit pour l'agrégation et le cryptage. Aussi, ce protocole ne consomme pas beaucoup d'énergie. L'inconvénient de ce protocole est qu'il n'assure pas le service d'intégrité des données qui est considéré comme un service très important dans la sécurité. On prend l'exemple suivant pour montrer que ce protocole n'assure pas l'intégrité, si un nœud p envoie sa donnée m cryptée avec la clé k à son parent q, $C = (m + k)\ mod\ M$, alors lorsque le nœud q est compromis il peut ajouter m' sans détection par le nœud **PUITS**, $(m' + C)\ mod\ M = (m' + m + k)\ mod\ M$.

3.3.3 Le protocole Elliptic Curve ElGamal

Contrairement aux protocoles décrits précédemment, ce protocole utilise un algorithme cryptographique à courbe elliptique ElGamal (**ECEG**) qui est une approche asymétrique. Cet algorithme ne consomme pas d'énergie comme les autres algorithmes à clé publique par exemple RSA.

L'algorithme de ECEG [31]

Paramètre :

Une clé privé x.

Une clé publique(G, H), G et H des points dans **ECEG**, $H = xG$.

Cryptage :

$C = [c_1, c_2] = [kG, kH + mG]$ = un point dans **ECEG**.

Décryptage :

$mG = (kH + mG) - x(kG)$.

Agrégation :

$\mathrm{C}_{12} = C_1 + C_2 = [(c_{11} + c_{21}), (c_{12} + c_{22})]$.

Ce protocole aussi n'assure pas le service de l'intégrité, et pour prouver cette remarque on prend l'exemple suivant : si un nœud p veut envoyer sa donnée à son parent q alors il envoie $C = [kG, kH + mG]$, dans le cas où q est un nœud compromis, il peut ajouter au C la valeur $[0, m'G]$ sans quelle soit détectée par le nœud **PUITS**, le nœud q envoie $[kG, kH + (m + m')G]$ à son parent.

Un autre inconvénient, est que ce protocole consomme beaucoup d'énergie par rapport aux protocoles précédents puisque il utilise un algorithme asymétrique.

3.3.4 Les inconvénients de cette approche

L'inconvénient principal de cette approche est le phénomène du rejet aveugle qui signifie que toutes les données reçues sont rejetées lorsque une seule donnée est corrompue. Le problème de rejet aveugle est posé dans ce type de protocole puisque la vérification des données est faite au niveau du nœud **PUITS**, qui connaît toutes les clés exigées pour la vérification. Ainsi, les données sont rejetées après leur arrivée au nœud **PUITS** et avoir traversé tout l'arbre de l'agrégation et avoir consommé par conséquent beaucoup d'énergie.

Un autre inconvénient, aussi important, de ces protocoles est qu'ils ne détectent que la valeur de l'agrégation finale corrompue, sans pouvoir localiser les noeuds compromis. Ces derniers sont capables de falsifier chaque fois la valeur de l'agrégation sans être localisés.

La perte des données pose aussi un autre problème. Dans le cas de perte de données, l'agrégation est appliquée sur les données d'un ensemble de nœuds feuilles. Et puisque le déchiffrement des données est fait au niveau du nœud **PUITS**, ce dernier trouve un problème pour choisir les clés adéquates afin de décrypter la valeur de l'agrégation de cet ensemble.

Les fonctions d'agrégation sont un autre inconvénient où cette approche est adéquate

pour une seule fonction d'agrégation qui est la somme. Si on veut calculer les fonctions comptage, moyenne et variance, cette approche consomme beaucoup d'énergie. De plus, il y a des fonctions d'agrégation qui ne sont pas réalisées par cette approche, comme maximum, minimum et autres.

3.4 Les protocoles basés sur le cryptage des données de proche-en-proche

Pour régler les problèmes de l'approche précédente, les chercheurs ont proposé cette approche, où les données dans ces protocoles sont cryptées par les nœuds qui captent l'information (sensing nodes) et décryptés par les nœuds qui font l'agrégation. Ces derniers calculent la valeur de l'agrégation et cryptent les résultats à nouveau et enfin le nœud **PUITS** récupère les résultats de l'agrégation finale.

Ces protocoles sont aussi divisés en deux catégories selon le type du réseau sur lequel ont été déployés : soit dans une architecteur plate, sans utilisation des clusters **DWSN (Distributed Wireless sensor Networks)**. Soit dans une architecture hiérarchique **HWSN (Hierarchical Wirless Sensor Networks)**.

3.4.1 Les protocoles implémentés dans DWSN

3.4.1.1 Le protocole SDA (Secure Data Aggregation using Commitment Schemes and Quasi Commutative)

1 Les hypothèses :

Le protocole **SDA** [24] assure la non répudiation et l'intégrité des données et de la requête en même temps en utilisant une fonction quasi commutative [1]. De plus, la confidentialité est assurée par un chiffrement symétrique en utilisant une clé partagée entre chaque nœud et son fils.

Contrairement aux protocoles présentés jusqu'à ici, le protocole **SDA** prend en considération un autre type d'attaque, où certains nœuds intermédiaires malhonnêtes peuvent falsifier la requête pendant la phase de propagation (diffusion de la requête). Ce protocole propose un schéma efficace, où les nœuds font des obligations de non répudiation sur leurs données et sur quelques propriétés de la requête.Par conséquent, la détection des nœuds malicieux qui veulent falsifier les résultats d'agrégation ou la requête devient plus facile. Le protocole **SDA** est conçu autour de ces suppositions :

– **SDA** assume un arbre hiérarchique simple.

1. La notion de quasi commutativité sera expliqué plus loin dans cette section

- La fiabilité de livraison des messages est assurée par d'autres protocoles de bas niveau.
- **SDA** utilise la fonction exponentielle modulaire $e_N(x, y) = x^y mod N$. Il est basé sur l'hypothèse que la fonction est à sens unique (facile à calculer, mais difficile à inverser). Un nombre entier N est rigide si $N = pq$ où $|p| = |q|$, p et q sont des nombres premiers forts (p est un nombre premier fort si $p = 2p' + 1$ tel que p' un nombre premier impair).Trouver le radical reste difficile du fait que les mathématiques n'ont pas progressé depuis 300 ans concernant la factorisation de très grands nombres en facteurs premiers. Le protocole **SDA** suppose que chaque nœud possède un nombre comme N, et un entier X pendant l'installation initiale.
- Le protocole **SDA** suppose que chaque nœud partage une clef cryptographique secrète avec son fils. Ainsi, toutes les communications entre un nœud et ses fils sont cryptées. Il suppose que chaque nœud P_i possède un nombre entier rigide N_i qui est utilisé par la fonction quasi-commutative comme il va être décrit dans les paragraphes qui suivent.
- la fonction d'agrégation est déterminée, distributive et ne dépend d'aucun ordre de lecture et elle est connue par tous les nœuds $f(d_1, d_2, d_3, ...d_n) = f(d_1, f(d_2, d_3, d_4), d_5, ..., d_n)$.

2 Les primitives cryptographiques utilisées :

On dit qu'une fonction h est quasi commutative si et seulement si :

$$\forall x \in X \; et \; y_1, y_2 \in Y \quad h(h(x, y_1), y_2) = h(h(x, y_2), y_1)$$

Dans le protocole **SDA**, chaque nœud p envoie une valeur v_p à son père, tel quev_p est une fonction quasi commutative définie comme suit :

$$v_p = X^{d_p \prod_{1 \leqslant i \leqslant k} d_i} \; mod \; N.$$

Où d_p est la lecture du noeud p, d_i est la lecture de son fils et X est une valeur définie au préalable.
v_p peut être évaluée comme suit :

$$v_p = (\cdots((X^{d1} mod N)^{d2} mod N)^{d3} mod N \cdots)^{dp} mod N.$$

Cette fonction est exécutée au niveau de chaque nœud, jusqu'à le nœud racine, qui doit trouver :

$$y = v_{racine} = X^{d_{racine} \prod_{1 \leqslant i \leqslant k} d_i} \; mod \; N, \; v_{racine} = X^W mod \; N \; et \; W = d_{racine} \prod_{1 \leqslant i \leqslant k} d_i$$

Où d_i c'est la lecture d'un noeud P_i dans le réseau et k le nombre de noeuds dans le réseau. Chaque noeud p_j maintient une fonction de hache partielle : $z_j = X^{w/dj} mod N$. Pendant la phase de vérification chaque nœud pj doit vérifier l'égalité suivante :

$$Y = z_j^{d_j} mod\ N.$$

3 Description du protocole SDA :

Le protocole **SDA** est exécuté en trois étapes après que l'arbre d'agrégation soit parcouru par la requête. Ces étapes sont la confirmation, la révélation et la vérification.

3.1 la phase de confirmation :

Pendant la phase confirmation, chaque nœud émet ses données et ne pourra pas les nier plus tard. De plus, il confirme quelques propriétés de la requête U_i reçue (par exemple, les spécifications de la fonction $f()$ qui doivent être calculées sur les données). Les données de chaque nœud ne sont pas révélées, puisque une fonction à sens unique difficile à inverser est utilisée, comme la fonction du résidu quadratique. Par conséquent, si la partie P_i veut confirmer ses données d_i, pendant la phase de confirmation chaque noeud calcule $C_i = d_i^2 mod N_i$, où $N_i = s_i t_i$ tels que si et ti sont des nombres premiers choisis par chaque nœud. Pendant cette phase, chaque nœud doit exécuter l'algorithme suivant :

L'algorithme de la phase de confirmation

v est un noeud dans l'arbre d'agrégation T

Debut

(1) **si** (v n'aura pas de fils dans T) alors

(2) envoie $G_v = C_v | U_v$ à son père

(3) **sinon**

(4) rassemble G_1, G_2,$\cdots$. G_k à partir de ses fils dans T

(5) calcule $R = X^{G_v \prod_{1 \leqslant i \leqslant k} G_i} modN$

(6) envoie R à son père

(7) **finsi**

Fin.

A la fin de cette phase, la racine de l'arbre d'agrégation aura la valeur . De plus, chaque nœud doit calculer la valeur : $z_j = X^{w/G_j} mod N$ qui sera utilisée plus tard pendant la phase de vérification.

3.2 La phase de révélation :

Pendant la phase de révélation, chaque nœud P_i révèle U_i, N_i et la donnée di à son père. De plus, chaque nœud calcule la fonction $f()$ sur les données d'un de ses fils qui doivent être envoyées à son père. Un autre arbre de validation est accumulé, cette fois au-dessus des valeurs de la fonction $f()$, comme montré ci-dessous.

L'algorithme de la phase révélation

v est un noeud dans l'arbre d'agrégation T

Debut

(1) **si** (v n'aura pas de fils dans T) **alors**
(2) envoie d_v, N_v et U_v à son père
(3) **sinon**
(4) envoie d_v, N_v et U_v à son père
(5) calcule $O_v = f(d_1, d_2, \cdots, d_k)$
(6) envoie O_v à son père
(7) **si**(les fils de v ne sont pas des feuilles) alors
(8) rassemble O_1,O_2,$\cdots$,O_k à partir de ses fils
(9) calcule $R' = X^{O_v \prod_{1 \leqslant i \leqslant k} O_i} modN$
(10) envoie R' à son père
(11) **finsi**
(12) **finsi**

Fin.

à la fin de la phase de révélation, la racine calcule la fonction d'agrégation finale $f()$.De plus, elle a deux arbres de validations, le premier est pour les données de chaque nœud, noté Y (pendant la phase de confirmation) et l'autre pour les résultats intermédiaires de la fonction $f()$, noté Y' tel que : $Y' = R' = X^{w'}$, où $W' = O_{racine} \prod_{1 \leqslant i \leqslant k} O_i$.
Chaque nœud calcule : $z_j = X^{w'/O_j} modN$ ou $z_j = X^{w'/d_j} modN$ selon que ce soit un nœud feuille ou pas.

3.3 La phase de vérification :

Pendant la phase de vérification, chaque nœud(P_l) vérifie l'exactitude de ses données ainsi que celle des données de ses fils.

Cette phase peut être réalisée en mettant en question chaque nœud fils de P_l, ce dernier détermine la donnée qu'il envoie pendant la phase de révélation (s'il est un nœud feuille), ou le résultat de la fonction exécutée au niveau de chaque nœud fils (s'il est un nœud non

feuille). Pendant cette phase chaque nœud doit exécuter l'algorithme suivant :

1^{er} **algorithme de la phase de vérification.**
Vérification des fils du noeud P_l
Debut
(1) **pour tout** P_r **fils de** P_l **faire**
(2) **si** (P_r est un noeud feuille) **alors**
(3) $C_r = d_r^2 mod N_r$
(4) Vérifie si $Z_r^{C_r|U_r} mod\ N = Y$
(5) **sinon**
(6) Vérifie si $(Z_r')^{O_r} mod\ N = Y'$
(7) **finsi**
(8) **fin pour**
Fin.

Après cette étape, le nœud P_l doit vérifier ses données et la valeur de sa fonction $f()$, comme indiqué dans l'algorithme ci-dessous :

2^{eme} **algorithme de la phase de vérification.**
Vérification du noeud P_l ses calculs.
Debut
(1) $C_l = {d_l}^2 mod N_l$
(2) **si** (P_l est un noeud feuille) **alors**
(3) Vérifie si $Z_l^{C_l|U_l} mod N = Y$
(4) **sinon**
(5) Vérifie **si** $Z_l^{C_l|U_l} mod N = Y$ **alors**
(6) **pour tout** P_r fils de P_l **faire**
(7) **si** (P_r est une noeud feuille) **alors**
(8) $Val_r = d_r$
(9) **sinon**
(10) $Val_r = O_r$
(11) **finsi**
(12) **fin pour**
(13) **finsi**
Fin.

4 Critiques du protocole SDA :

Quelques inconvénients du protocole **SDA** sont présentés dans ce qui suit :

- **Les attaques possibles sur SDA :**
  SDA peut détecter n'importe quel type d'attaque soit dans l'étape de propagation de requête ou dans l'étape d'agrégation des données en utilisant la fonction quasi commutative. Le problème avec **SDA**, c'est qu'il ne peut détecter que l'attaque sur la donnée agrégée ou la requête mais il ne peut pas localiser exactement les nœuds malicieux.
- **Le coût :**
  Le protocole **SDA** assume que la fiabilité des messages est assurée par des protocoles de bas niveau, cette supposition gaspille l'énergie et la bande passante puisque chaque message perdu doit être retransmis pour garder la fiabilité de livraison. Ainsi, le coût de calcul est vraiment très important puisque dans les 3 étapes du protocole, chaque nœud doit faire des calculs exponentiels modulaires qui gaspillent l'énergie de manière considérable. Dans les coûts de communications, on constate que le protocole envoie un nombre important des messages, pendant les trois phases afin de valider une seule valeur agrégée.
- **Le temps de réponse du protocole :**
  Le protocole **SDA** assume que la fiabilité des messages est assurée par des protocoles de bas niveau, cette supposition fait que chaque message perdu doit être retransmis pour garder la fiabilité de livraison qui augmente le temps de réponse de manière significative. Pour que le nœud **PUITS** valide la valeur d'agrégation finale, il doit attendre 3 phases avec des calculs exponentiels modulaires qui peuvent augmenter le temps de réponse.
- **Le rejet aveugle :**
  Ce protocole souffre du problème du rejet aveugle, car la vérification des données est faite après les phases de confirmation et révélation où ces deux phases consomment beaucoup d'énergie et transfèrent beaucoup de messages. Alors, lorsqu'un seul nœud est compromis dans le réseau la valeur finale de l'agrégation sera rejetée après le transfert de beaucoup des messages et la consommation de beaucoup d'énergie.
- **L'attaque d'usurpation d'identité :**
  Ce protocole n'utilise aucune authentification (**MAC**) entre le nœud fils et le nœud père. Alors un nœud voisin au nœud père peut lui envoyer des paquets en utilisant l'adresse source d'un autre nœud fils, sans qu'il soit détecté par le nœud fils.

3.4.1.2 Le protocole SDAP (Secure Data Aggregation without Persistent Cryptographic Operations in Wireless Sensor Networks)

1 Hypothèse :

Le protocole **SDAP** [28] cible les adversaires qui veulent détruire les informations produites à partir du réseau de capteurs. Il ne gère pas la confidentialité des paquets de données. **SDAP** suppose qu'un nœud ne peut pas envoyer des messages dont l'adresse *IP* source est une adresse d'un autre nœud voisin. Cette supposition est facile à réaliser, puisque chaque nœud peut écouter ses voisins lorsqu'ils envoient des messages, et peut facilement détecter n'importe quel message contenant son adresse comme adresse source. **SDAP** suppose aussi que la communication entre deux nœuds P_1 et P_2 est sécurisée, cette solution est réalisable grâce à des protocoles de distribution de clés. Cette dernière hypothèse ne signifie pas que **SDAP** utilise toujours des messages cryptés et authentifiés. Ils sont utilisés seulement si un comportement anormal est détecté.

2 Construction de l'arbre d'agrégation sécurisé SAT (Secure Aggregation Tree) :

SDAP est construit sur une structure d'arbre où chaque nœud peut contrôler le comportement de son père. Comme tous les protocoles étudiés dans ce mémoire, **SDAP** ne peut pas détecter le compromis des nœuds feuilles.

Pour permettre à un nœud fils de contrôler son père, il a besoin de recevoir tous les messages reçus ou transmis par son père (les messages reçus à partir de ses frères, et les messages transmis à partir de son père à son grand père). Le nœud fils calcule la valeur agrégée et il la compare avec la valeur agrégée envoyée à partir de son père.

3 Terminologie :

nb_hop indique le nombre de sauts entre chaque nœud et le nœud **PUITS**. Initialement, *nb_hop* transmis dans les messages est égal à zéro et au niveau des nœuds est égal à l'infinie. Chaque fois qu'un nœud reçoit le message d'invitation, il garde le chemin et la valeur de *nb_hop* minimale.

Une *clique* est définie comme l'ensemble des nœuds frères avec leur père, tel que chaque nœud dans la clique peut écouter tous les messages transmis à partir de son père à son grand père ainsi que tous les messages transmis à partir de ses frères à son père.

4 L'algorithme distribué pour construire SAT :

SDAP propose un algorithme distribué pour construire l'arbre d'agrégation sécurisé **SAT** (*Secure Aggregation Tree*). La construction du **SAT** comprend 4 étapes :

– **Étape 1 :**

Le nœud **PUITS** diffuse localement un message d'invitation pour les nœuds voisins immédiats (un seul saut). Ce message inclut l'*IDs* de tous les nœuds qui peuvent devenir ses fils.

– **Étape 2 :**

Une fois qu'un nœud reçoit le message d'invitation, si ce nœud ne s'est pas encore joint à l'arbre d'agrégation et le message d'invitation inclut ce nœud comme nœud fils, alors ce nœud se joint à l'arbre d'agrégation et enregistre l'émetteur de ce message comme nœud père. De plus, il diffuse sur un saut le message *JOIN* pour informer ses voisins de cette décision. Lorsque un nœud P_1 appartenant à l'arbre d'agrégation reçoit un message d'invitation (*JION*) avec *nb_hop* plus petit que *nb_hop* enregistré localement, il l'enregistre pour l'utiliser lorsque le père courant est compromis pour le choix du nouveau chemin.

– **Étape 3 :**

Après qu'un nœud P_1 est attaché à l'arbre d'agrégation, il vérifie ses voisins à un et à deux sauts, par l'envoi du message d'invitation, il cherche toutes les cliques qui peuvent appartenir à P_1. Chaque nœud ayant reçu ce message peut comparer ses voisins avec le champ *IDs* inclus dans le message. Si P_1 ne trouve pas des cliques adéquates, il devient un nœud feuille. Autrement, P_1 sélectionne la clique maximale et diffuse localement un message d'invitation où *nb_hop* est incrémenté par 1. Tous les nœuds dans la clique sélectionnée deviennent des fils de P_1.

– **Étape 4 :**

Répéter les étapes 2 et 3 jusqu'à ce qu'aucun nœud ne joint l'arbre. Il est possible que des nœuds ne peuvent pas joindre l'arbre d'agrégation pourtant il existe des chemins qui mènent vers le nœud **PUITS**. On appelle ces nœuds les nœuds *éparpillés*. Les nœuds voisins d'un nœud éparpillé sont aussi des nœuds éparpillés. Dans les réseaux denses, le nombre de nœuds éparpillés est négligeable par rapport au nombre de nœuds total.

En fin d'exécution de **SAT** chaque nœud doit enregistrer les informations suivantes :

– *IDs* de ses voisins d'un et deux *sauts*.
– *ID* de son père dans l'arbre d'agrégation.
– *IDs* de ses frères dans l'arbre d'agrégation.

5 Détection des nœuds trompant dans l'agrégation de données :

Election pondérée :

En raison de la contrainte de la topologie dans **SAT**, chaque nœud peut contrôler tous les messages reçus ou envoyés à partir de son père. Tout d'abord, **SDAP** utilise tous les messages de contrôle dans l'élection pondérée et le processus de rétablissement local doit être chiffré et authentifié avec quelques mécanismes de communication sécurisée. Cela doit garantir que la décision finale est correcte, une fois que la mauvaise conduite potentielle

est détectée. Aussi, nous soulignons de nouveau que la communication sécurisée est exigée seulement quand un nœud d'agrégation travaille incorrectement. Une fois qu'un nœud capteur détecte que son père peut être compromis, il envoie un message d'alerte à tous ses voisins sauf le nœud père.

Il faut faire la différence entre un message d'alerte et un message de confirmation de détection (*detection-confirmation*). Un message d'alerte signifie qu'il y a probablement un nœud compromis. Par contre, un message de confirmation de détection signifie qu'il y a une confirmation finale de l'existence d'un nœud compromis. Le message d'alerte doit être chiffré et authentifié. Un message d'alerte doit inclure les informations suivantes :

- *ID* du nœud trompant.
- *ID* du nœud détectant.
- La valeur de confiance.

La valeur de confiance indique la probabilité qu'un nœud est compromis. Plus la valeur de confiance est grande plus la probabilité que le nœud trompant est compromis.

Chaque nœud qui reçoit le message d'alerte vérifie si le nœud trompant est son père (P : le nœud père), si oui il calcule la valeur du poids de confiance en utilisant la formule suivante :

$$F = \frac{\sum_{i=1}^{m1} f_i}{m}$$

Où :

f_i : La valeur de confiance incluse dans le message d'alerte d'un noeud frère i ;

m : Le nombre total des noeuds frères ;

m_1 : Le nombre des noeuds frères qui ont envoyé le message d'alerte.

Si la valeur du poids de confiance F est plus grande qu'un certain seuil le nœud père p est considéré comme un nœud compromis, et le message detection-confirmation est diffusé avec un TTL limité. Chaque nœud q recevant un tel message, vérifie si le nœud p est son fils, si c'est le cas il ignore tous les messages qu'il pourrait recevoir à partir de p. Autrement, si le nœud p est son père alors il sélectionne un autre candidat qui devient son père en se basant sur les messages d'invitation déjà enregistrés. Noter que le *nb_hop* déjà enregistré dans les messages d'invitation est plus petit que le *nb_hop* déjà enregistré dans le nœud q, donc il est impossible de former des routes fermées.

S'il n'existe aucun candidat pour devenir le père de q alors le nœud q devient un nœud éparpillé.

Les nœuds éparpillés actuellement ont des chemins qui mènent vers le nœud **PUITS**, sauf que ces chemins ne sont pas sécurisés puisque ils contiennent des nœuds compromis. Mais lorsque le nœud **PUITS** a besoin des données de ces nœuds (données non agrégées), il

peut utiliser des mécanismes pour sécuriser ces chemins par les signatures et le chiffrement des données.

6 Critique du protocole SDAP :

Le protocole SDAP présente quelque, inconvénients, parmi les quels :

- **Le temps de réponse du protocole :** Le fonctionnement de **SDAP** est que chaque nœud envoie ses données bruts à son père, ce dernier agrège les données reçus puis envoie à son tour la donnée agrégée à son père sans utiliser aucun algorithme cryptographique, pour cette raison le temps de réponse est efficace sur **SDAP** sauf pour les nœuds éparpillés ou chaque message envoyé au nœud **PUITS** doit être chiffré et signée.
- **Les attaques possibles sur SDAP :**
  On distingue deux types d'attaques, en premier l'attaque ordinaire où le nœud malicieux veut casser la sécurité sans connaître les clés d'un nœud, en second l'attaque d'un nœud compromis où l'adversaire reprogramme un nœud dans le réseau. Ce nœud compromis connaît toutes les clés physiques d'un nœud victime.
  - **L'attaque ordinaire (sans compromission) :**

  Le protocole **SDAP** peut détecter n'importe quelle attaque de ce type. Puisque l'architecture de **SAT** permet que chaque nœud puisse contrôler tous les messages reçus ou envoyés à partir de son père, donc chaque nœud P_1 peut détecter n'importe quel message d'un autre nœud qui contient son adresse comme adresse source.
  - **L'attaque d'un nœud compromis :**

  Comme tous les protocoles précédents, **SDAP** ne peut détecter la compromission des nœuds feuilles. **SDAP** est construit sur une structure d'arbre où chaque nœud peut contrôler le comportement de son père. Ce dernier est considéré comme nœud compromis si la valeur du poids de confiance est plus grande qu'un certain seuil. La valeur du poids de confiance dépend du nombre des nœuds frères qui ont envoyé le message d'alerte, donc **SDAP** ne peut pas détecter la collusion de plusieurs nœuds compromis dans une clique lorsque le nombre de ces nœuds est plus grand qu'un certain seuil (ces nœuds n'envoient aucune alerte).
- **L'analyse du coût :**
  On fait l'analyse du coût sur les deux types de nœuds, les nœuds ordinaux et les nœuds éparpillés :
  - **Les nœuds ordinaux :**

  L'avantage de **SDAP** est que dans les situations normales **SDAP** n'utilise aucun algorithme cryptographique ce qui minimise la consommation de l'énergie de manière considérable. L'inconvénient de **SDAP** est que chaque nœud doit stocker, les *IDs* de

ses voisins à un et à deux sauts et l'*ID* de son père dans l'arbre d'agrégation, ainsi que les *IDs* de ses frères dans l'arbre d'agrégation, ce qui peut gaspiller la mémoire dans les nœuds capteurs. De plus, chaque nœud doit écouter tous les messages reçus ou envoyés à partir de son père qui peut gaspiller l'énergie dans les nœuds capteurs.

- **Les nœuds éparpillés :**

Malgré que le nombre des nœuds éparpillés est négligeable par rapport au nombre total des nœuds dans le réseau entier, lorsque le nœud **PUITS** veut récupérer les données de ces nœuds, chaque nœud éparpillé doit chiffrer et signer ses données qui peuvent gaspiller l'énergie dans le réseau de capteurs de manière considérable.

– **L'attaque d'usurpation d'identité :**

Malgré que **SDAP** suppose qu'un nœud p ne peut pas envoyer des messages dont l'adresse source est une adresse d'un autre nœud voisin q, mais il se peut que p ne soit pas voisin au nœud q et soit voisin au parent de q, alors le nœud p peut utiliser l'identité du nœud q sans qu'il soit détecté par ce dernier.

– **Possibilité d'implémentation :**

Ce protocole suppose que l'architecture dans laquelle le protocole est déployé est un ensemble de graphes fortement connexes chevauchés entre eux. Cette architecture est difficile à réaliser. D'après les simulations qu'on a effectué nous avons remarqué que presque 50% des nœuds deviennent des nœuds éparpillés, par conséquence ce protocole devient similaire au protocole de routage sécurisé simple où chaque nœud envoie ses données cryptées vers le nœud **PUITS**.

3.4.2 Les protocoles implémentés dans HWSN

3.4.2.1 Le protocole SRDA (Secure Reference-Based Data Aggregation)

1 les hypothèses

Le protocole **SRDA** [21] assure l'authentification mutuelle entre les nœuds en utilisant un protocole de défi réponse. La confidentialité et l'intégrité sont assurées par un chiffrement symétrique (RC6). **SRDA** est conçu autour des suppositions suivantes :

– Le protocole **SRDA** est implémenté au dessus d'un protocole d'agrégation de données hiérarchique (ex : *LEACH, PEGASIS...*).

– Chaque nœud capteur doit stocker deux ensembles (trousseaux) de clés. Chaque trousseau comporte un ensemble de paires (identificateur de clé, clé). L'un de ces trousseaux est local de taille x (assure l'interconnexion inter cluster) et l'autre est global de taille y (assure l'interconnexion intra cluster). Par conséquent, chaque nœud doit stocker $x + y$ clés.

– La distribution de ces trousseaux est effectuée de façon à ce que deux nœuds voisins,

P_1 et P_2 doivent partager une clé commune.

L'architecture du réseau est un ensemble de clusters. En effet, la communication entre les nœuds d'un même cluster et de clusters différents est réalisée en utilisant respectivement les trousseaux locaux et globaux.

2 Description du protocole :

Etant donné que la communication est un consommateur important de l'énergie dans un nœud capteur, il est important de réduire le nombre de bits transmis. Dans les algorithmes conventionnels d'agrégation de données, les nœuds capteurs transmettent leurs données brutes au chef du cluster (leader). Comme les données dans chaque paquet peuvent être les mêmes, ceci conduit à un gaspillage d'énergie et de bande passante. Cependant, **SRDA** transmet les données différentielles plutôt que les données brutes captées. La différentielle est calculée par rapport à une donnée de référence. Par exemple, dans un réseau de températures où le chef du cluster considère la valeur 40° comme une température de référence, lorsqu'un nœud capteur capte une mesure de température qui est 42°, la valeur envoyée sera la différentielle 2° (42°- 40°) au lieu de 42°. Par conséquent, l'agrégation différentielle permet de diminuer considérablement la quantité de données transmises du nœud capteur vers le chef du cluster et par conséquence l'énergie consommée. L'avantage de base derrière l'agrégation des données différentielles est que les mesures captées ne changent que lorsqu'un événement important se produit (par exemple, un incendie pour un réseau de température). Généralement les événements importants se produisent moins fréquemment que les événements ordinaires.

SRDA utilise *SRDA_ASP ()* comme protocole d'agrégation de données. Ce protocole sera présenté ultérieurement dans les prochaines sections. Les étapes principales de SRDA sont récapitulées dans ce qui suit :

2.1 Les étapes principales de SRDA () :

- Étape 1 : Pour chaque session de données, exécuter les étapes 2, 3...6.
- Étape 2 : Un nœud capteur Pcalcule sa valeur de référenceM_1en calculant la moyenne des N dernières valeurs de données captées ($N \geq 1$).
- Étape 3 : Le premier paquet est envoyé avec la valeur M_1 au chef du cluster. Celui-ci considère la valeur M_1 comme étant la valeur de référence du nœud P. Les paquets de session sont chiffrés en utilisant *SRDA_ASP ()*.
- Étape 4 : Le chef du cluster crée une entrée de référence pour le nœud P avec la valeur M_1.
- Étape 5 : Pour les lectures suivantes M_j, $j \geq 2$, le nœud capteur envoie seulement la différentielle c-à-d $(M_j - M_1)$au lieu de la donnée brute M_j.

– Étape 6 : Quand une session de données d'un nœud capteur est terminée, le chef du cluster enlève l'entrée de référence correspondante à cette session.

Dans **SRDA**, la période d'une session est la durée de communication établie entre le nœud capteur et le chef du cluster. **SRDA** est indépendant du schéma de clustering et peut être appliqué sur n'importe quel algorithme de clustering. Il peut être appliqué à tous les niveaux de la hiérarchie du cluster. L'efficacité de cette technique est plus importante lorsque la valeur de référence est plus grande que la valeur différentielle. Un autre facteur important pour la performance de cette technique est la divergence des données captées. Lorsque la divergence devient plus petite les gains réalisés par l'agrégation différentielle de données augmentent puisque, un petit nombre de bits suffit pour représenter les valeurs différentielles de données. Le stockage et la consultation des valeurs de références effectuées par les chefs des clusters sont atténués puisque, les valeurs de référence sont maintenues seulement pendant la session de transfert de données.

Une variante de **SRDA** met à jour la valeur de référence avec l'arrivée de chaque nouveau paquet. Dans ce cas, la donnée différentielle transmise est $(M_j - M_{j-1})$.

2.3 Le protocole de sécurité d'agrégation de SRDA

Contrairement aux autres réseaux sans fil, l'importance des paquets circulant dans les réseaux de capteurs sans fil varie selon la proximité de ces paquets au nœud **PUITS**. Considérant des fonctions typiques telle que *Max()* ou *Count()*, les paquets transmis par les chefs des clusters les plus hauts dans l'hiérarchie de cluster au nœud **PUITS** représentent un résumé d'un grand nombre de paquets des niveaux inférieurs. En se basant sur cette observation, le niveau de la sécurité du réseau est graduellement renforcé ce qui permet d'économiser graduellement l'énergie. L'augmentation du niveau de sécurité peut être mise en œuvre en utilisant différents algorithmes cryptographiques ou un même algorithme avec des paramètres réglables. Cependant, les contraintes mémoire des nœuds capteurs limitent l'utilisation de plusieurs algorithmes cryptographiques. Donc, le choix d'un algorithme avec paramètres réglables semble le plus approprié aux réseaux de capteurs sans fil. Un exemple d'algorithmes cryptographiques avec de telles propriétés est celui du chiffrement par bloc **RC6** [23]. **RC6** est un algorithme paramétré où la taille du bloc, de la clé et le nombre des rounds sont variables. pour être sécurisé est égal à 10, et le nombre actuel de rounds à exécuter est égal à 12, alors la marge de sécurité est de 20 %. Pour économiser l'énergie, **SRDA** utilise des marges de sécurité plus petites pour les chefs des clusters de niveau inférieur en comparaison avec les chefs des clusters des niveaux les plus hauts.

Les étapes de base du protocole de sécurité d'agrégation **SRDA** qui s'exécute au niveau de chaque nœud sont décrites ci-dessous.

2.4 Protocole : SRDA_ASP ()

- Etape 1. Découvrir la distance du nœud **PUITS**, c'est-à-dire le nombre de sauts (h).
- Etape 2. Calculer la marge de sécurité (S).

$$S = (1/h) * 100$$

- Etape 3. Augmenter le nombre minimum de rounds de RC6 Par S%.
- Etape 4. Chiffrer les données utilisant RC6 avec le nombre de rounds trouvés dans l'étape 3.

L'idée de *SRDA_ASP* est de fournir différents niveaux de sécurité pour différentes classes de données et cela en vue de garantir une transmission de données sécurisée et une utilisation optimale d'énergie.

3 Critiques du protocole SRDA :

Quelques inconvénients du protocole **SRDA** sont présentés dans ce que suit :

- **L'attaque d'un nœud compromis :**
  SRDA est vulnérable vis-à-vis d'un nœud compromis, par exemple un adversaire qui récupère les clés physiques d'un chef du cluster P peut introduire un nouveau nœud qui doit jouer le rôle de P. Ce nouveau nœud falsifie la valeur d'agrégation de plusieurs nœuds sans que l'utilisateur final puisse le détecter. Cette vulnérabilité rend la sécurité de **SRDA** très faible, vue la probabilité élevée de cette attaque dans les réseaux de capteurs qui sont généralement déployés dans des environnements non surveillés.
- **Le passage à l'échelle :**
  SRDA repose sur des protocoles de gestion de clés qui utilisent des trousseaux de clés pour l'établissement des liens sécurisés. La taille de ces trousseaux est proportionnelle à la taille du réseau. La taille du réseau est donc strictement limitée par la capacité de stockage des nœuds capteurs.
- **Le coût :**
  Ce protocole est basé sur l'idée de crypter les données qui ont une taille minimale de 8 octets, et toutes les communications sont cryptées. Par conséquence, ce protocole consomme beaucoup d'énergie.
- **La non intégrité de l'agrégation :**
  Ce protocole n'assure pas l'intégrité de l'agrégation, car lorsque le chef de cluster est compromis ceci peut falsifier la valeur de l'agrégation de son cluster sans qu'elle ne soit detctée par les membres de ce cluster ou le nœud **PUITS**.

3.4.2.2 Le protocole SecureDAV : A Secure Data Aggregation and Verification Protocol for Sensor Networks

1 Les hypothèses :

Le protocole **SecureDAV** [25] assure l'authentification et l'intégrité des données en utilisant le schéma de signature à seuil et l'arbre de Merkle [27]. De plus, la confidentialité est assurée par un chiffrement symétrique.
Les anciens protocoles basés sur les protocoles de données centralisées (*LEACH, PEGASIS,...*) supposent qu'un adversaire peut introduire un certain nombre de nœuds compromis. Néanmoins, un attaquant peut exécuter une grande variété des attaques. Il peut compromettre le chef du cluster. Le protocole **SecureDAV** [25] peut continuer son fonctionnement normal avec un nombre de nœuds compromis allant jusqu'à t capteurs avec $t < n/2$ capteurs où n est le nombre de capteurs dans un cluster. Pendant la phase d'initialisation, chaque nœud doit enregistrer les paramètres du domaine *ECC (Elliptic curve cryptography)*, abordé plus loin, et la clé *EC_public* (clé publique) du noeud **PUITS**. Une fois le réseau est déployé, chaque nœud calcule sa paire de clés privées et *EC_public*. Cette dernière doit être diffusée sur tous les nœuds du cluster. La paire de clés est utilisée pour sécuriser les communications entre les nœuds capteurs (génération d'une clé x_{ij} secrète entre deux nœuds i et j en utilisant *Diffie-Hellman*).

2 Les primitives cryptographiques utilisées :

SecureDAV utilise le principe de cryptographie basé sur les courbes elliptiques, où les clés employées pour un chiffrement par courbe elliptique sont plus courtes qu'avec un système basé sur le problème de la factorisation comme **RSA**. De plus, *l'ECC* procure un niveau de sécurité équivalent ou supérieur aux autres méthodes. La résistance d'un système fondé sur les courbes elliptiques repose sur le problème du logarithme discret dans le groupe correspondant à la courbe elliptique.
L'exemple suivant illustre la méthode de changement des clés en utilisant la courbe elliptique.

3 Le protocole d'établissement de clé de cluster CKE (Cluster Key Establishment) :

3.1 Objectif du protocole :

CKE est un protocole qui permet d'établir une clé secrète au niveau de chaque cluster, de telle sorte que chaque nœud capteur doit avoir une partie de la clé secrète. Chaque partie est utilisée pour générer une signature partielle en utilisant *EC-DSA* [15]. Le chef du cluster, rassemble toutes les signatures partielles à partir des nœuds membres. De

plus, il combine ces signatures dans une seule signature puis envoie le message signé au noeud **PUITS**. Cette dernière connaît déjà la clé publique peut donc vérifier la validité du message.

3.2 Description du protocole CKE :

3.2.1 Génération des secrets partiels de cluster :

1.)

   - **a)** chaque noeud P_i choisit deux polynômes aléatoires sur un domaine elliptique de degré t :
     $f_i(z) = a_{i0} + a_{i1}z + a_{i2}z^2 + ... + a_{it}z^t$, $f_i^{'}(z) = b_{i0} + b_{i1}z + b_{i2}z^2 + ... + b_{it}z^t$.
     Le noeud P_i pose $z_i = a_{i0} = f_i(0)$. De plus,il calcule :
     - $C_{ik} = (a_{ik} + b_{ik})T$ pour $k = 0, 1, ...t$.
     - Les parties $s_{ij} = f_i(j)\ mod\ p, s_{ij}^{'} = f_i^{'}(j)\ mod\ p$. pour j = 1,2, $\cdots$, n.
   - **b)** P_i diffuse C_{ik}. De plus, il envoie s_{ij} et $s_{ij}^{'}$ à P_j par un canal sécurisé en utilisant la clé x_i^j.
   - **c)** Chaque noeud P_j vérifie les parties reçues :
     Pour i=1,2,$\cdots$,n P_j vérifie si :

     $$(s_{ij} + s_{ij}^{'})T = \sum_{k=0}^{t} j^k C_{ik}\ mod\ p \qquad (equation\ 1).$$

     Si le noeud P_j détecte que l'équation 1 n'est pas vérifiée pour un noeud i, il diffuse un message d'erreur contre P_i.
   - **d)** Lorsque le noeud P_i reçoit cette erreur à partir de P_j, il diffuse les valeurs s_{ij} et s_{ij}' qui ont satisfait l'équation 1.
   - **e)** Un noeud P_i classe un autre noeud P_j comme étant un noeud disqualifier si :
     - Il reçoit plus de t messages d'erreur contre P_j à partir de noeuds différents.
     - La réponse du noeud P_j dans l'étape 1.d, est une réponse incorrecte.

2.) Chaque noeud P_i construit un ensemble des noeuds non-disqualifiés appelé QUAL.

3.) Chaque noeud P_i calcule sa partie de la clé secrète de cluster $x_i = \sum_{j \in QUAL} S_{ji}\ mod\ p$.
   La clé secrète qu'il ne peut pas calculer par n'importe qu'elle noeud est $x = \sum_{j \in QUAL} z_i\ mod\ p$

3.2.2 Génération de clé publique pour la clé secrète de cluster

1.) Chaque noeud $P_i \in QUAL$ expose $y_i = z_i T$:
   - a) P_i diffuse $A_{ik} = a_{ik} T$ pour k=1,$\cdots$t.
   - b) Chaque noeud P_j vérifie les valeurs diffusées à partir des noeuds de QUAL. P_j vérifie si :

$$s_{ij} T = \sum_{k=0}^{t} j^k A_{ik} \ mod \ p \qquad (equation\ 2).$$

   Si la vérification a échoué pour un noeud P_i, P_j diffuse un message d'erreur contre P_i. Ce message contient les valeurs s_{ij} et s'_{ij} qui ont satisfait l'équation 1 mais ne satisfont pas l'équation 2.

2.) Pour un noeud P_i qui reçoit un message d'erreur valide, les autres noeuds répètent la création à partir du début et calcule z_i, $f_i(z)$, A_{ik} pour k=0,$\cdots$.t.

3.) Calcul de la clé publique y :
   Pour chaque noeud dans QUAL, pose $y_i = A_{i0} T = z_i T$. Le noeud **PUITS** calcul

$$y = \sum_{i \in QUAL} y_i \ mod \ p$$

   On a $y = xT$ puisque :

$$y = \sum_{i \in QUAL} y_i \ mod \ p \ = \sum_{i \in QUAL} z_i T \ mod \ p \ = (\sum_{i \in QUAL} z_i \ mod \ p \ )T \ mod \ p \ = \ xT.$$

4 Le protocole de sécurité d'agrégation des données et vérification de protocole :

Après l'exécution du protocole **CKE**, chaque nœud membre du cluster doit avoir une partie de la clé secrète. Cette dernière est utilisée pour générer une signature partielle sur les données déjà lues. Chaque chef de cluster agrège les données reçues à partir des nœuds membres en calculant la moyenne sur ces données. Cette dernière doit être diffusée sur tous les membres du cluster, puis chaque nœud capteur compare sa lecture avec la moyenne, si la différence est moins d'un seuil, il crée une signature partielle sur la moyenne en utilisant la partie de la clé secrète. De plus, il envoie la signature partielle au chef du cluster qui combine les différentes signatures dans une seule signature qu'elle doit être envoyée avec la moyenne au noeud **PUITS**.

La validité de cette signature est vérifiée en utilisant la clé publique déjà enregistrée au niveau du noeud **PUITS**. Un attaquant ne peut pas générer la clé du cluster, pour réussir il doit injecter dans le réseau plus de t nœuds compromis.

L'algorithme du protocole de sécurité d'agrégation des données et protocole de vérification :

Notation :

Notation	**Description**
CH_i	Le chef du cluster i
$\|CH_i\|$	Le nombre de capteurs dans un cluster i
k_j^i	La clé secrète partielle du nœud$j$dans le cluster i
x_m^j	La clé secrète partagée entre un nœud m avec j qui peut être un nœud ordinaire ou chef du cluster ou noeud **PUITS**
$h()$	Une fonction de hache de faible collision
$ENC(x, M)$	Cryptage symétrique du message$M$ en utilisant la clé partagée x
$Sign(k, M)$	Une signature partielle sur le message M
$Send(src, dest, M)$	Primitive de communication pour l'envoi du message M à partir de src vers de dest
$Bcast(src, M)$	Primitive de *broadcast* du message M à partir de source *src*.

TABLE 3.1: Notations

Le protocole de sécurité d'agrégation des données et protocole de vérification

Debut

(1) /* le nœud capteur transmet ses lectures cryptées et hachées
(2) pour assurer la confidentialité, l'authentification et l'intégrité */
(3) **pour** $j = 1$ **à** $|CH_i|$ **faire**
(4) **début**
(5) $Encrypted := ENC(x_j^{CH_i}, R_j)$;
(6) $Send(j, CH_i, Encrypted \mid h(R_j))$;
(7) **fin pour**
(9) /* le chef du cluster agrège les lectures */
(10) $avg_i = 0$;
(11) **pour** chaque capteur j dans le cluster i **faire**
(12) $avg_i := avg_i + (R_j/|CH_i|)$;
(13) /* le chef du cluster diffuse la moyenne des lectures aux membres du cluster*/
(14) $Bcast(CH_i, avg_i)$;
(15) /* génération des signatures partielles et combinées vers la signature générale*/
(16) **pour** $j = 1$ à $|CH_i|$ faire
(17) **début**
(18) $Partialsignature_j := sign(k_j^i, h(avg_i))$;
(19) $Send(j, CH_i, Partialsignature_j)$;
(20) **fin pour**
(21) CH_i combine les signatures partielles vers $combine_i$;
(22) /* transmet l'agrégation des lectures avec la signature au noeud **PUITS** */
(23) $Send(CH_i,\ S, ENC(x_{Chi}^S, avg_i)||combine_i)$;
(24) Le noeud PUITS vérifie $combine_i$, en utilisant la clé publique
(25) /* assurer l'authentification et l'intégrité */

Fin.

4.2 Intégrité des lectures :

Le schéma de signature à seuil assure l'authenticité et l'intégrité de la valeur de la moyenne. Mais l'intégrité des lectures est assurée à l'aide de l'arbre de hache de Merkle. Les capteurs transmettent leurs lectures chiffrées avec leurs haches au chef du cluster. Ce dernier crée l'arbre de hache de Merkle basé sur les valeurs hachées reçues dans les messages. Lorsque le noeud **PUITS** reçoit la valeur de la moyenne cryptée et signée, elle vérifie la signature en utilisant la clé publique.

Le chef du cluster compromis peut envoyer la valeur moyenne fausse signée correc-

tement. Pour éviter cette situation, le noeud **PUITS** peut demander au chef du cluster d'envoyer les différentes lectures avec les haches de Merkle. La Figure 3.2 illustre le principe de fonctionnement de l'arbre de hache de Merkle où chaque $r_i = ENC(x_i^S, R_i)$. Par exemple, pour authentifier la lecture R_3, le chef du cluster envoie R_3 avec M_4, M_{12}, M_{56} et M_{16} au noeud **PUITS**. Cette dernière calcule r_3 en utilisant x_i^S. Elle authentifie R_3 en utilisant M_{16} (envoyé par le chef du cluster) et calcule aussi $h(h(h(h(r_3)||M_4)||M_{12})||M_{56})$, qui doit être comparée avec M_{16}.

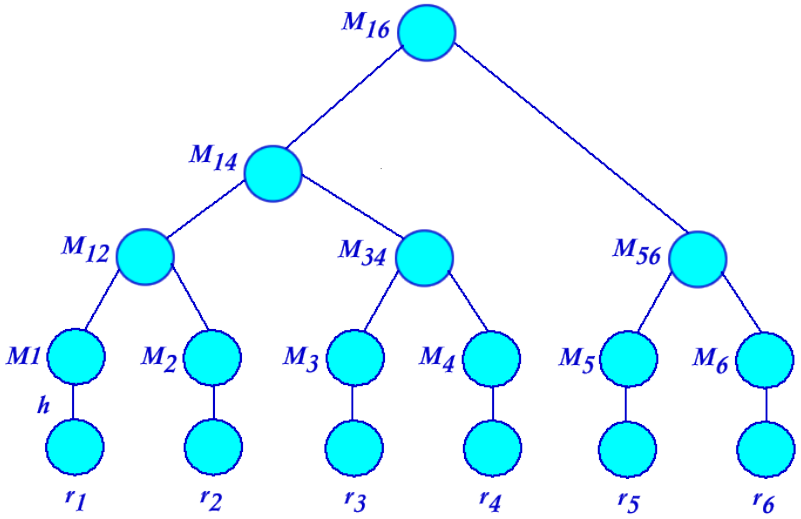


FIGURE 3.2: Exemple d'utilisation de haches de Merkle

5 Critiques du protocole SecureDAV :

Quelques inconvénients du protocole **SecureDAV** sont présentés dans ce qui suit :

– **Le temps de réponse du protocole :**

D'après [25], la taille des clés d'un algorithme elliptique est égale à 2 fois la taille des clés d'un algorithme symétrique. Ainsi, le temps de réponse des protocoles basés sur la courbe elliptique est plus élevé par rapport aux algorithmes symétriques. Par exemple, au moment où le temps de réponse d'un algorithme symétrique (Skipjack) est égal à 0.26 ms, on remarque que le temps de réponse d'un algorithme elliptique soit égal à 2.19 s [38]

Donc, l'utilisation de ce type d'algorithmes influe sur le temps de réponse du protocole.

Après l'exécution de l'algorithme CKE, chaque nœud doit envoyer sa lecture cryptée en utilisant sa clé commune avec le chef du cluster (Diffie-Hellman). Ce dernier calcule la moyenne de toutes les lectures reçues, qui doit être diffusée à tous les membres. Puis, chaque nœud P_i compare sa lecture avec la moyenne, si la différence

est moins d'un seuil, P_i crée une signature sur la moyenne en utilisant la partie de la clé secrète puis il envoie la signature partielle au chef du cluster qui combine les différentes signatures sur une seule signature qu'elle doit être envoyée avec la moyenne au noeud **PUITS**.

Le noeud **PUITS** peut demander au chef du cluster d'envoyer les différentes lectures de différents nœuds avec les haches de Merkle.

Ces opérations peuvent influer sur le temps de réponse dans le cas où le nombre de nœuds est important dans les clusters.

- **Le coût :**

  Pour sécuriser les lectures et la valeur de la moyenne il faut générer et diffuser beaucoup de messages au niveau de chaque nœud qui peut gaspiller l'énergie de ces nœuds.

  Chaque nœud doit stocker les paramètres du domaine et deux polynômes. De plus, chaque chef de cluster doit stocker l'arbre de Merkle qui peut gaspiller une ressource rare dans les nœuds capteurs, c'est la mémoire.

- **Le passage à l'échelle :**

  Dans **SecureDAV** chaque chef du cluster envoie la valeur de la moyenne au noeud **PUITS**. Donc, la taille du réseau dépend de la portée du chef de cluster. Par conséquent, **SecureDAV** ne permet pas le passage à l'échelle.

  De plus, lorsque le nombre de nœuds dans un cluster est important, alors l'arbre de *Merkle* stocké dans le chef du cluster devient très important. Le chef du cluster pour vérifier l'intégrité des lectures doit envoyer les lectures de différents membres chiffrés et authentifiés au noeud **PUITS** qui peut gaspiller rapidement ses ressources.

3.5 Conclusion

On a abordé dans ce chapitre les différents protocoles de sécurité d'agrégation des données dans les réseaux de capteurs, avec plus de détail sur le principe de fonctionnement, et les types des algorithmes cryptographiques utilisés. De plus, une partie illustre les critiques des différents protocoles.

On remarque que tous les protocoles basés sur le cryptage des données de bout-en-bout ont deux problèmes principaux, c'est le problème de rejet aveugle et la non localisation des nœuds malicieux dans le réseau. Ces problèmes impossibles à résoudre, puisque sont liés au type de vérification de ces protocoles, où la vérification dans ce type des protocoles se fait au niveau du noeud **PUITS**. Dans la deuxième approche la vérification se fait d'une manière totalement distribuée et instantanée, pour permettre le passage à l'échelle, la localisation des nœuds malicieux et éviter le problème du rejet aveugle.

4

SEDAN : Secure and Efficient protocol for Data Aggregation in wireless sensor Networks

4.1 Introduction

La fonction principale des nœuds capteurs est de rassembler des données à partir de l'environnement, et de collaborer avec les autres nœuds pour router ces données vers un centre de traitement appelé le nœud **PUITS**. Ces nœuds sont généralement équipés d'une batterie faible et d'une unité de stockage très limitée. Par conséquence, n'import quel protocole dans ce type de réseaux doit prendre en considération ces contraintes [2].

D'un autre coté, pour assurer la tolérance aux pannes et la qualité de captage dans le réseau de capteurs, une solution permettant d'augmenter la redondance des données consiste à déployer ce réseau avec une grande densité. Malheureusement, cette redondance va augmenter la consommation d'énergie dans les nœuds et le risque de collisions dans le réseau.

Pour éviter ces problèmes, plusieurs solutions, utilisant l'agrégation de données dans le réseau ont été proposées pour agréger les données captées. Ces agrégations sont basées sur des fonctions mathématiques : comme **MAX**, **MIN**, **SUM**,..etc. Le paradigme de l'agrégation des données est une clé essentielle pour augmenter la durée de vie du réseau de capteurs, en réduisant le nombre de diffusions et de collisions. D'autre part, l'agrégation des données est un point de vulnérabilité potentiel pour les attaquants. Ces derniers

peuvent injecter des fausses données ou carrément détruire la valeur de l'agrégation.

Plusieurs solutions ont été proposées [21], [24], [30] et [31]. Ces solutions peuvent être divisées en deux catégories : *proche-en-proche* et *bout-en-bout*. Dans l'approche des protocoles *proche-en-proche*, la vérification de l'intégrité se fait dans le réseau ainsi qu'au niveau du **PUITS**. Par contre, dans les protocoles bout-en-bout, le seul responsable de la vérification est le nœud **PUITS**.

Les principaux inconvénients de ces solutions sont la centralisation, au niveau du nœud **PUITS**, de la vérification et le rejet aveugle. La vérification centralisée peut s'effectuer de manière totale ou partielle. Par conséquence, l'application n'est pas totalement distribuée, et la vérification se fait seulement lorsque toutes les données agrégées arrivent au nœud **PUITS**. Ainsi, ces protocoles ne permettent pas le passage à l'échelle. Le deuxième problème est le rejet aveugle cela veut dire que lors de la détection d'un nœud malicieux, le nœud **PUITS** rejette toutes les valeurs de l'agrégation déjà reçues. Ceci est la cause de perte des données correctes.

Dans ce mémoire, nous présentons un nouveau protocole de sécurité de l'agrégation. Il est basé sur la vérification à deux sauts inspirée du protocole **SAWN (Secure Aggregation for wireless Network)** [20]. La nouveauté est qu'il élimine les problèmes décrits précédemment (la centralisation de la vérification et le rejet aveugle). Dans notre solution nommée, **SEDAN (Secure and Efficient Data Aggregation protocol for wireless sensor Networks)**[33], chaque nœud vérifie instantanément les données envoyées par ses voisins à deux sauts, de plus, il vérifie les valeurs d'agrégation envoyées par ses voisins immédiats. Cette amélioration élimine la transmission des fausses données, et optimise la consommation de l'énergie. L'idée de notre protocole est l'utilisation d'un nouveau type de clés appelée clé symétrique à deux sauts. Ce nouveau type de clés, permet le partage d'une clé entre chaque deux nœuds voisins à deux sauts, mais elle est inconnue par les voisins immédiats. Cette méthode permet à n'importe quel nœud de vérifier l'intégrité des données envoyées par ses voisins à deux sauts, et assure que les données ne seront pas modifiées par ses voisins immédiats.

4.2 Notations et terminologie

Dans le tableau suivant nous introduisons les différentes notations et terminologies utilisées dans la description des protocoles **SAWN** et **SEDAN**.

Notation	**Description**
$A, B, C, \ldots$	Nœuds capteurs
ID_A	Identificateur du nœud A
S	Nœud PUITS
d_A	Donnée du nœud A, elle peut représenter une lecture ou une valeur d'agrégation.
N_A	Nonce généré par le nœud A.
f	Fonction d'agrégation.
$MAC(K, m)$	Authentification du message m en utilisant la clé K.
$E(K, m)$	Cryptage du message m en utilisant la clé K.
$A \rightarrow$B : m	A envoie le message m à B.
$K_{A,B}$	Clé symétrique partagée entre le nœud A et le nœud B. A et B peuvent être voisins d'un seul saut ou voisins de deux sauts.
$M_1 \|\| M_2$	Concaténation de message M_1 et M_2.
MK_A	Clé Master du nœud A.
G	Fonction à sens unique.
$P(A)$	Parent du nœud A.
$GP(A)$	Grand parent du nœud A.
$MAC(K, M)$	Authentification du message M en utilisant la clé K.
K_{AS}	Clé unique partagée entre le nœud A et le nœud PUITS.
R_A	Valeur lue par le nœud A
K_{Ai}	La i^{eme} clé pour le nœud $A = E(K_{AS}, i)$

4.3 Le protocole SAWN (Secure Aggregation for Wireless Networks)

Le protocole **SAWN** [20] cible les adversaires qui veulent détruire les informations produites à partir du réseau de capteurs (i.e. attaques actives). Il ne gère pas la confidentialité des données. C'est le premier protocole qui a introduit le mécanisme de vérification à deux sauts.

SAWN exploite l'idée de retarder l'agrégation . Effectivement, pour un nœud x, l'agré-

gation des données de ses fils ne se fait pas à son niveau, mais elle s'effectue au niveau de son père. Cela augmente le coût de transmission, mais permet de garantir l'intégrité pour les réseaux où deux nœuds consécutifs ne sont pas compromis. Pour l'authentification des paquets diffusés par le nœud **PUITS**, il n'est pas pratique d'utiliser un algorithme asymétrique. Le protocole **SAWN** adopte le protocole $\mu TESLA$[26].

Le protocole **SAWN** est conçu autour de ces suppositions :

- Le nœud **PUITS** est puissant et diffuse directement des messages à tous les nœuds.
- Les nœuds capteurs à basse puissance communiquent seulement avec leurs voisins.
- La fiabilité de livraison des messages est assurée par d'autres protocoles de bas niveau.
- Avant le déploiement, chaque nœud doit partager des clés secrètes avec le nœud **PUITS**.
- Le réseau ne doit pas avoir deux nœuds compromis consécutifs.

4.3.1 Description du protocole :

La Figure 4.1 illustre une partie du réseau de capteurs composé de huit nœuds (A, B, C...H) plus un nœud **PUITS** puissant. Le nœud **PUITS** rassemble les informations de tous les nœuds capteurs et les transmet vers la station de base en utilisant un protocole de routage. Chaque nœud feuille transmet un message à son père. Ce message inclut la lecture des données du nœud, son *id* ainsi qu'un **MAC** calculé grâce à la clé K_{Ai}. Cette dernière est partagée entre le nœud A et le nœud **PUITS**. Initialement la clé K_{Ai} n'est pas connue par les autres capteurs. Le nœud père stocke le message ainsi que son **MAC** jusqu'à la révélation de la clé K_{Ai} par le nœud **PUITS**. Ainsi, il vérifiera le **MAC** et envoie une alarme en cas de différence.

Le protocole **SAWN** implique des étapes séparées pour l'envoi des données vers le nœud **PUITS**. Pour décrire ce protocole, on prend l'exemple illustré dans la Figure 4.1. Les nœuds A, B, E et F envoient leurs lectures au nœud **PUITS** via l'arbre montré dans la figure 4.1.

- Les nœuds feuilles envoient des messages à leur père. Les messages incluent des **MACs** calculés avec la clé temporaire. Par exemple :

$$A \rightarrow C : ID_A, d_A, MAC(K_{Ai}, d_A)$$

Chaque clé est utilisée par un seul message et une seule fois pour empêcher l'attaque du *rejeu*.

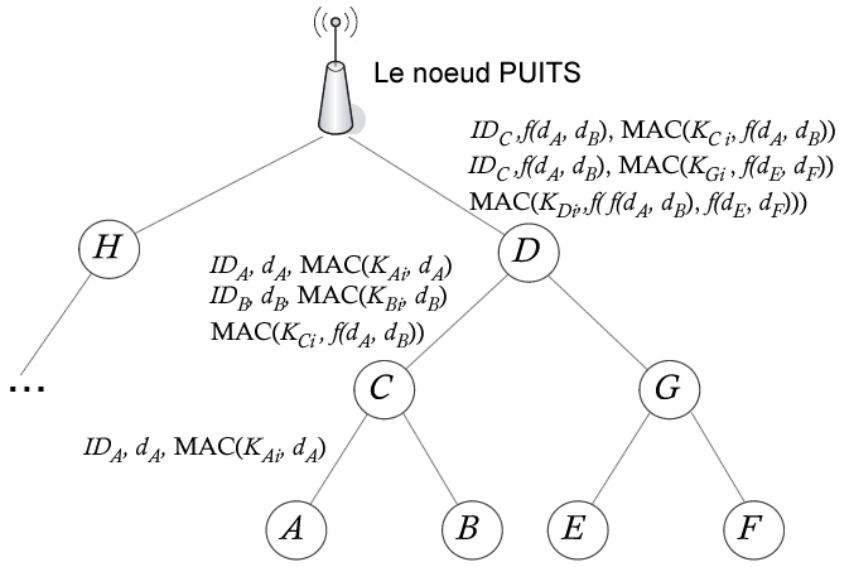


FIGURE 4.1: Le processus d'agrégation dans SAWN

– Les nœuds intermédiaires reçoivent les messages de leurs fils. Á la réception d'un tel message, un nœud intermédiaire ne peut pas vérifier le MAC de son fils car la clé de ce dernier ne sera révélée que pendant la phase de vérification. Pour le moment, le père stocke juste le message, le **MAC** et il attend les paquets des fils. Ensuite il envoie un message à son père contenant les lectures de ses fils, leurs **MACs**, ainsi que le **MAC** calculé sur la valeur d'agrégation.

  Il n'y a aucun besoin de transmettre la valeur d'agrégation calculée, puisque C peut calculer $f(R_A, R_B)$ depuis les valeurs R_A et R_B. Aussi, il n'est pas nécessaire de transmettre l'ID_C à D, étant donné que D connaît la topologie du réseau ainsi il peut déterminer l'émetteur du message.

$$\begin{aligned} C \rightarrow D : \quad & ID_A, d_A, MAC(K_{Ai}, d_A) \\ & ID_B, d_B, MAC(K_{Bi}, d_B) \\ & MAC(K_{Ci}, f(d_A, d_B)) \end{aligned}$$

– Le nœud D reçoit les messages des noeuds C et G. Pour chacun d'eux, D calcule les valeurs de l'agrégation des lectures de ses petits-fils c'est-à-dire (A, B, E et F). Il transmet alors les valeurs agrégées de ses petits-fils, l'ID de ses fils et leurs valeurs de **MAC**. D calcule aussi et transmet le **MAC** de la valeur d'agrégation suivante : $f(R_A, R_B, R_E, R_F) = f(f(R_A, R_B), f(R_E, R_F))$. Puisque la fonction de l'agrégation est connue à tous les nœuds, le **MAC** calculé par C authentifiera la valeur calculée par D. Les lectures des capteurs et les valeurs de **MAC**s reçues à partir de C et G

sont stockées pour la vérification postérieure.

$$\begin{aligned} D \rightarrow S : \quad & ID_C, f(d_A, d_B), MAC(K_{Ci}, f(d_A, d_B)) \\ & ID_G, f(d_E, d_F), MAC(K_{Gi}, f(d_E, d_F)) \\ & MAC(K_{Di}, f(f(d_A, d_B), f(d_E, d_F)) \end{aligned}$$

– Le nœud **PUITS** reçoit le message de D. Il peut calculer la valeur de l'agrégation finale, $f(R_A, R_B, R_C, R_D)$ en utilisant $f(R_A, R_B)$ et $f(R_C, R_D)$.

Puisque le nœud **PUITS** a une clé partagée temporaire avec chaque nœud capteur (par exemple K_{Di}), il peut vérifier si le message reçu dans le pas final a été transmis par D en calculant le **MAC** de l'agrégation. Cela valide que D a envoyé le message final, mais ne peut pas valider qu'il a correctement reflété la lecture des autres nœuds. Le nœud **PUITS** reçoit aussi les **MAC**s et les lectures de ses petits-fils et peut connaître leurs valeurs. Le but du protocole **SAWN** est d'authentifier toutes les lectures qui ont participé à la valeur d'agrégation, sans la nécessité que chaque lecture doit être envoyée au nœud **PUITS**.

Pour valider des données, le nœud **PUITS** révèle les clés temporaires des nœuds, en émettant un seul message, authentifié par la clé courante μ**TESLA**. Pour assurer ce but, **SAWN** suppose que le nœud **PUITS** soit puissant, d'où, il est capable de joindre le réseau entier par un seul saut. Cette révélation des clés permet à chaque nœud la vérification de l'intégrité des données de ses petits fils et les valeurs de l'agrégation de ses fils.

Les nœuds capteurs et le nœud **PUITS** passent à la clé temporelle suivante, en utilisant un compteur i déjà synchronisé et une clé partagée entre chaque nœud et le nœud **PUITS** (EX : K_{AS}). La nouvelle clé est égale à ($K_{Ai} = E(K_{AS}, i)$).
Le nœud **PUITS** émet toutes les clés temporaires des nœuds. Cependant, chaque nœud capteur a besoin seulement de quelques unes d'entre elles. Par exemple, le nœud C a besoin de K_{Ai} pour vérifier **MAC**(K_{Ai}, R_A). Si les émissions des clés sont synchronisées, il n'est pas nécessaire d'écouter toutes les émissions des clés.

Si un nœud détecte un message erroné dans l'étape de validation de données, il envoie un message d'alarme. Des alarmes sont émises par un parent quand il détecte que le **MAC** d'agrégation d'un fils est contradictoire avec les données des petits-fils, ou bien quand les **MAC** de données eux-mêmes sont erronés.

4.3.2 Critiques du protocole SAWN :

– **L'attaque d'un nœud compromis :**
Le protocole **SAWN** détecte tous les nœuds compromis non consécutifs qui tentent d'introduire des données fausses dans le réseau.
SAWN peut trouver des difficultés dans les cas suivants :

- Une attaque où un nœud compromis peut éliminer certains messages de ses fils. Pour résoudre ce problème, la technique du *Watch-Dog* est utilisée dans le réseau, où le nœud fils P écoute son père lorsqu' il envoie les messages à son grand père, dans les cas où les messages n'incluent pas les données de P, ce dernier envoie au nœud **PUITS** une alerte contre son père en utilisant un autre chemin.
- Le protocole **SAWN** ne peut pas détecter deux nœuds consécutifs compromis (un nœud père et son fils).

– **L'analyse du coût :**

L'analyse de coût est basée sur le coût de communication (l'énergie de transmission et l'énergie de l'écoute), puisque ce sont les consommateurs les plus importants dans un nœud capteur.

Pour faire notre analyse, on considère un arbre de routage idéal où le nombre de sauts entre chaque nœud feuille et le nœud **PUITS** est égal à d. De plus chaque nœud doit avoir b fils.

Pour la premiére étape, b^d nœuds feuilles envoient leurs lectures à leurs parents. On utilise m pour représenter la longueur en bits d'une lecture et c la longueur en bits de *ID* et **MAC**. Dans l'étape suivante, chaque nœud envoie les lectures, les valeurs de **MAC** plus le **MAC** de l'agrégation. On fait l'hypothèse que la taille de la valeur d'agrégation est égale aux tailles des valeurs de lectures. On peut citer comme exemple de fonctions qui permettent cela : **MAX**, **MIN**, **AVERAGE**. Donc, chaque nœud père a b fils. Ainsi il transmet $b(m+c)+cbits$. Dans le niveau secondaire on a $b^{d-1}bits$ nœuds, donc le nombre total de bits qui doivent être transmis dans le seconde niveau est $b^d(m+c)+b^{d-1}cbits$. Aussi, dans l'étape suivante chaque nœud envoie $b(m+c)+cbits$. Enfin, le nombre total des bits transmis est :

$$F = \frac{m}{b-1}(2b^{d+1} - b^d - b^2) \; + \; \frac{c}{b-1}(2b^{d+1} - b^2 - b) \;\; bits$$

Par contre, le nombre de bits transmis dans l'agrégation non sécurisée est : $m(\frac{b^{d+1}-b}{b-1})$ $bits$. Lorsqu'on fait une application numérique où $m = 22$ $octets$, $c = 8$ $octets$, $b = 4$ et $d = 5$, on remarque dans le cas du protocole **SAWN**, le nombre total des bits transmis est 82.5 $koctets$. Cependant, dans l'agrégation non sécurisée, le nombre total est de 32 $koctets$. Dans cet exemple on remarque bien que l'agrégation dans **SAWN** génère un nombre triple de messages par rapport à l'agrégation non sécurisée.

Le coût de calcul et le gaspillage de la mémoire sont négligeables par rapport à la communication, chaque nœud intermédiaire doit calculer la valeur d'agrégation de ses petits-fils, donc il doit calculer b fonctions d'agrégation. Ainsi, chaque nœud intermédiaire doit authentifier à son tour, ses fils et ses petits-fils après que la clé

soit révélée par le nœud **PUITS**, donc il calcule $b^2 + b\ MAC$ ($b^2\ MAC$ pour ses petits-fils et $b\ MAC$ pour ses fils).
Chaque nœud doit stocker les messages reçus à partir de ses fils jusqu'à la révélation des par le nœud **PUITS**, d'où la nécessité d'espace mémoire important. Ces messages contiennent les données de ses fils et petits-fils, ainsi dans le premier niveau, chaque nœud doit stocker $(m + c)b\ bits$ et à partir du second niveau, chaque nœud doit stocker $(m + c)b^2 + cb\ bits$.

– **Le passage à l'échelle :** Cette solution ne permet pas le passage à l'échelle pour un grand réseau qui peut contenir des milliers de nœuds. En effet, le nœud **PUITS** diffuse les clés révélées, vers tous les nœuds pendant la phase de vérification. Cette diffusion cause deux problèmes : un retard significatif de détection des nœuds compromis, et la taille du réseau qui dépend de la portée du nœud **PUITS**. En outre, la diffusion de ces clés d'authentification pour tout le réseau exige l'utilisation de plusieurs paquets.

Pour analyser la consommation de l'énergie, on considère le scénario suivant :
La révélation des clés des nœuds exige 6 octets dont 2 octets pour l'identificateur et 4 pour la clé. Aussi, chaque paquet doit contenir un **MAC** (4 octet) généré en utilisant le protocole de clé $\mu TESLA$. Si on considère qu'on utilise le simulateur **TOSSIM** [39], la taille de paquet dans ce simulateur est 29 octets par défaut. Par la suite, chaque paquet peut contenir seulement les clés de 4 nœuds.
La figure 4.2 illustre la variation de nombre de messages diffusés en fonction de la taille des clés transmises. On remarque que la solution **SAWN** transmet un nombre important de paquets, ce dernier augmente selon la taille du réseau $O(n)$.
De plus, la phase de vérification ne peut se lancer que si toutes les clés sont révélées. Dans un réseau important, cette phase peut engendrer un retard important, car le nœud **PUITS** doit attendre un certain temps pour valider les données reçues.

– **Le rejet aveugle :** Un autre inconvénient majeur de **SAWN** est le nombre des données rejetées. Dans **SAWN**, et autres [24], [29], [30], [31], la violation de l'intégrité des données dans n'importe quel emplacement dans le réseau, oblige le nœud **PUITS** à rejeter la valeur de l'agrégation reçue, alors que cette dernière représente une vue sur la totalité de la branche. Ce rejet peut produire une perte des données significative.

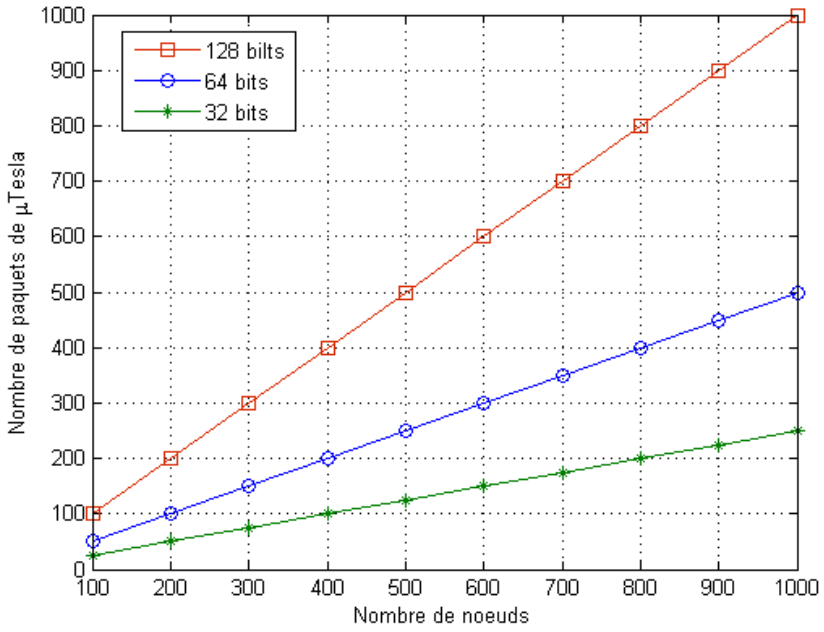


FIGURE 4.2: Nombre de paquets de μTESLA exigés vs Nombre de nœuds

4.4 Notre solution : (Secure and Efficient protocol for Data Aggregation in wireless sensor Networks)

4.4.1 Le modèle général

Pour maîtriser les contraintes décrites précédemment, notre solution surmonte les besoins de diffusion des clés par le nœud **PUITS**, en introduisant un nouveau type de clés appelé :

clé symétrique à deux sauts. Ces clés sont partagées entre les voisins à deux sauts et cachée par rapport aux voisins immédiats. Lorsqu'un nœud transmet sa donnée ou sa valeur d'agrégation, il calcule aussi un **MAC** en utilisant sa clé *symétrique à deux sauts* partagé avec son grand père. Du fait que cette clé est cachée au nœud père, l'intégrité des données est assurée, et chaque modification de la valeur de l'agrégation peut être détectée par le grand père. Ainsi, puisque la vérification dans **SEDAN** est distribuée dans le réseau, elle facilite le passage à l'échelle.

Du moment que **SEDAN** bloque (sans aucun retard), les données erronées au niveau des nœuds compromis, il évite l'arrivée de ces données au niveau du nœud **PUITS** et empêche ces données de falsifier les autres données correctes. Par conséquence toute valeur d'agrégation arrivant au nœud **PUITS** est correcte et peut être validées immédiatement.

4.4.2 Les hypothèses

– la fonction d'agrégation doit vérifier la relation suivante :

$$f(d_1, d_2, d_3, \ldots, d_n) = f(d_1, f(d_2, d_3, d_4), d_5, \ldots, d_n).$$

C'est-à-dire que le résultat final peut être calculé par n'importe quelle combinaison des sous résultats en utilisant la même fonction. Il existe plusieurs fonctions d'agrégation qui vérifient cette propriété comme *MIN, MAX, AVREAGE,...* etc.

– Dans la description de **SEDAN**, on suppose que la topologie du réseau est sous forme d'un arbre. Il faut noter que notre protocole peut être utilisé dans n'importe quelle structure hiérarchique.

– On suppose que l'arbre ne contient pas deux nœuds compromis consécutifs.

– On suppose aussi que l'attaquant, pour compromettre un nœud, dure un délai T_{min}.

4.4.3 Description du protocole

SEDAN est un protocole de sécurité léger construit selon les étapes suivantes :

1. **L'établissement des clés :**
   L'étape initiale consiste à l'établissement de toutes les clés *symétriques*. Elle est exécutée une seule fois pendant la durée de vie du réseau, en contraste du protocole d'agrégation. D'autre part, pour renforcer la sécurité, les clés doivent être régénérées périodiquement.

   Chaque nœud a besoin de générer quatre types de clés *symétriques* :
   – Une clé *symétrique* d'un seul saut partagé avec son père *(one-hop_pair-wise)*.
   – Une clé *symétrique* d'un seul saut partagé avec son fils *(one-hop_pair-wise)*.
   – Une clé *symétrique* à deux sauts partagés avec son grand père *(tow-hop_pair-wise)*.
   – Une clé *symétrique* à deux sauts partagés avec son petit fils (*tow-hop_pair-wise*).

   Le détail de l'établissement de ces clés est donné dans la section 4.5.

2. **L'authentification des données :**
   Lorsqu'un nœud X veut envoyer sa donnée d_X, il envoie à son père le paquet suivant :

$$\begin{aligned} &ID_X, N_X, d_X, \\ &MAC(K_{X,P(X)}, N_X \mid\mid d_X), \\ &MAC(K_{X,GP(X)}, d_X \mid\mid N_X) \end{aligned}$$

   Tous simplement, le nonce N_X est un numéro de séquence qui permet d'éviter l'attaque du rejeu de même donnée. La première **MAC** est appelée *One Hop MAC (OHM)*, elle est calculée en utilisant la clé *symétrique* d'un seul saut partagé avec

le pére, et permet à ce dernier de vérifier l'origine des paquets. Cette vérification empêche l'attaque d'usurpation d'identité. La seconde **MAC** est appelée *Tow Hops MAC (THM)*, elle est calculée en utilisant la clé *symétrique* à deux sauts partagée avec le grand père. Le *THM* permet l'exécution de mécanisme de vérification à deux sauts, comme décrit dans l'étape cinq.

3. **La vérification de l'intégrité des données d'un seul saut :**
   Lorsque un nœud Y reçoit un paquet de données à partir de son fils, il vérifie le *OHM* pour valider l'origine du paquet, et sauvegarde le reste des informations : ID_{child}, N_{child}, d_{child} et THM_{child}.
4. **L'authentification de la valeur de l'agrégation :**
   Lorsque le nœud Y reçoit les données de tous ses fils, il calcule la valeur de l'agrégation de ces données. Par conséquence, $f(d_{child1}, \ldots, d_{childn})$ représente la donnée du nœud Y. Ce dernier doit calculer sa *OHM* et sa *THM* respectivement.

   $$MAC(K_{Y,p(Y)}, N_Y||f(d_{child_1}, \ldots, d_{child_n}))$$
   $$MAC(K_{Y,GP(Y)}, f(d_{child_1}, \ldots, d_{child_n})||N_Y)$$

   Donc, le nœud Y transmet à son père :

   $$Id_{child_1}, N_{child_1}, d_{child_1}, THM_{child_1}$$
   $$\vdots$$
   $$Id_{child_n}, N_{child_n}, d_{child_n}, THM_{child_n}$$
   $$Id_Y, N_Y, MAC(K_{Y,p(Y)}, N_Y||f(d_{child_1}, \ldots, d_{child_n}))$$
   $$MAC(K_{Y,GP(Y)}, f(d_{child_1}, \ldots, d_{child_n})||N_Y)$$

5. **La vérification de l'intégrité des données de deux sauts :**
   Lorsque un nœud Z reçoit un paquet d'agrégation à partir du nœud Y. Le nœud Z doit vérifier le comportement de l'agrégation du nœud Y.
   - Pour garantir que les données envoyées par les petits fils sont correctes, le nœud Z vérifie *THM* de chacun d'eux. De plus, le nœud Z reconstruit la valeur de l'agrégation correcte du nœud Y en utilisant les données des petits fils.
   - Le nœud Z ayant la valeur de l'agrégation correcte, peut comparer avec la valeur de l'agrégation calculée par le nœud Y. Donc, **SEDAN** peut détecter les valeurs de l'agrégation erronée immédiatement sans aucun retard, et toutes les données erronées sont arrêtées pour préserver les valeurs de l'agrégation des branches correctes.

   Le nœud Z considère la valeur de l'agrégation de chaque nœud fils comme une donnée de ce nœud, et répète l'étape 4 en envoyant :

   $$ID_{child_1}, N_{child_1}, d_{child_1}, THM_{child_1}$$

$$\vdots$$
$$ID_{child_n}, N_{child_n}, d_{child_n}, THM_{child_n}$$
$$ID_Z, N_Z,$$
$$MAC(K_{Z,P(Z)}, N_Z \mid\mid f(d_{child_1}, \ldots, d_{child_n}))$$
$$MAC(K_{Z,GP(Z)}, f(d_{child_1}, \ldots, d_{child_n}) \mid\mid N_Z)$$

Remarque : si le nœud agrégateur Y, veut envoyer sa lecture avec le paquet de la valeur de l'agrégation, il doit transmettre sa lecture avec un extra $OHM = MAC(K_{Y,P(Y)}, N_Y||d_Y)$. Le nœud Y envoie aussi son THM calculé sur la valeur de l'agrégation et cette dernière, à son tour, est calculée sur sa lecture et les données de ses fils.

$$ID_{child_1}, N_{child_1}, d_{child_1}, THM_{child_1}$$
$$\vdots$$
$$ID_{child_n}, N_{child_n}, d_{child_n}, THM_{child_n}$$
$$ID_Y, N_Y, d_Y, MAC(K_{Y,P(Y)}, N_Y \mid\mid d_Y)$$
$$MAC(K_{Y,P(Y)}, N_Y \mid\mid f(d_{child_1}, \ldots, d_{child_n}, d_Y))$$
$$MAC(K_{Y,GP(Y)}, f(d_{child_1}, \ldots, d_{child_n}, d_Y) \mid\mid N_Y)$$

Le saut suivant Z vérifie les $THMs$ de tous ses petits fils et OHM du nœud Y. Alors, le nœud Z peut calculer la valeur de l'agrégation du nœud Y et renvoie les paquets exigés (voir étape 5). Alors, la lecture du nœud Y devient comme une lecture d'un fils de Y.

4.4.4 Exemple :

La Figure 4.3 illustre les messages transférés pendant la phase de l'agrégation en utilisant notre protocole **SEDAN.**

– Le nœud A envoie, au nœud C, sa lecture, son OHM et son THM

$$\begin{aligned} A \rightarrow C : \quad & ID_A, N_A, d_A, \\ & MAC(K_{A,C}, N_A \mid\mid d_A) \\ & MAC(K_{A,D}, d_A \mid\mid N_A) \end{aligned}$$

– Le nœud B fait la même opération, le nœud C reçoit les deux paquets des données. C vérifie l'origine des paquets reçus et agrège tous les lectures et envoie le paquet suivant :

$$\begin{aligned} C \rightarrow D : \quad & ID_A, N_A, d_A, MAC(K_{A,D}, d_A \mid\mid N_A) \\ & ID_B, N_B, d_B, MAC(K_{B,D}, d_B \mid\mid N_B) \\ & ID_C, N_C, MAC(K_{C,D}, N_C \mid\mid f(d_A, d_B)) \\ & MAC(K_{C,S}, f(d_A, d_B) \mid\mid N_C) \end{aligned}$$

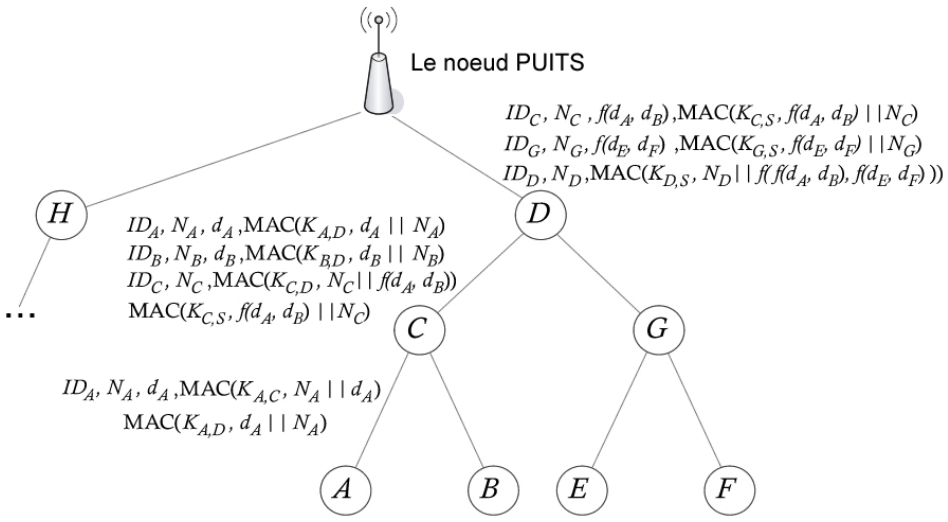


FIGURE 4.3: Le processus d'agrégation dans SEDAN

– Le nœud D peut vérifier l'origine des données de A et B, et calcule aussi la valeur de l'agrégation de C. Le même mécanisme est appliqué sur les messages envoyés par le nœud G.

$$
\begin{aligned}
D \rightarrow S : \quad & ID_C, N_C, f(d_A, d_B), \\
& MAC(K_{C,S}, f(d_A, d_B) \,||\, N_C) \\
& ID_G, N_G, f(d_E, d_F) \\
& MAC(K_{G,S}, f(d_E, d_F) \,||\, N_G) \\
& ID_D, N_D, \\
& MAC(K_{D,S}, N_D \,||\, f(f(d_A, d_B), f(d_E, d_F)))
\end{aligned}
$$

Remarque : On remarque que le dernier message *THM* du nœud D n'est pas envoyé, puisque c'est le dernier nœud voisin au nœud **PUITS**, dans ce cas *THM* n'a plus d'intérêt.

4.5 Le protocole EPKE (EXTENDED PAIR-WISE KEY ESTABLISHMENT)

Il existe plusieurs travaux [35], [36], [22], [37], qui traitent le problème de l'établissement d'une infrastructure pour sécuriser les communications dans les réseaux de capteurs. Mais le schéma le plus adéquat, est celui qui utilise des clés symétriques et aussi doit être lui même organisé sans connaître les informations sur le déploiement.

L'absence d'une infrastructure fixe, et les limites des ressources en termes de stockage et de calcul des nœuds capteurs, mettent les protocoles de prédistribution comme les plus appropriés pour les réseaux de capteurs. Dans ces protocoles, tous les nœuds sont préchargés par un secret informatique, ce dernier est utilisé pour l'établissement des clés

entre les nœuds voisins.

Dans [22], le protocole de gestion de clés basé sur le mécanisme de prédistribution, repose sur une approche probabiliste. Chaque nœud porte une liste de k clés distinctes. Cette liste est choisie de manière aléatoire à partir d'un large ensemble de clés. Deux nœuds peuvent établir une clé *symétrique* entre eux, si et seulement s'ils peuvent trouver une clé commune dans leurs listes. L'inconvénient de cette approche de protocoles est l'exigence d'un espace mémoire important pour stocker les listes des clés dans le cas où la taille du réseau est importante.

Les auteurs dans [35], [36] proposent une famille de protocoles permettant l'établissement des clés *symétriques* en se basant sur le concept *Transitory Initial Key* **(TIK)**. Dans ces protocoles, la même clé transitoire initiale est préconfigurée dans chaque nœud. Ce dernier utilise cette clé pour générer des clés *symétriques* partagées avec ses voisins. Après la phase d'installation, chaque nœud supprime la clé initiale de sa mémoire *EEPROME*. La supposition principale du concept **TIK** est que l'attaquant ne peut pas obtenir la clé initiale en compromettant un nœud légitime pendant la phase de l'installation. Le temps d'installation est supposé court et représente *le temps minimum* (T_{min}) requis par le nœud pour l'établissement des clés avec ses voisins.

Dans cette section, on présente une solution pour l'établissement des clés *symétriques* d'un et deux sauts, en se basant sur le schéma d'installation des clés initiales transitoires **(TIK)** proposé par *LEAP* [35] et *OTMK* [36].

4.5.1 Initialisation

Avant le déploiement, les capteurs sont préchargés par une clé initiale transitoire K_{IN}. Chaque nœud U peut dériver à partir de cette clé sa clé master MK_U :

$$MK_U = G(K_{IN}, ID_U)$$

Lorsque le nœud est déployé dans le réseau, la clé initiale est utilisée pour l'établissement de clé symétrique avec chaque voisin. Chaque clé initiale est valide seulement pour un temps T_{min}. Après ce temps, chaque nœud supprime la clé initiale K_{IN}.

4.5.2 L'établissement de clé symétrique d'un seul saut

Ce type de clés est utilisé dans **SEDAN** pour calculer *OHM*. Ce dernier permet de créer un lien sécurisé entre deux nœuds voisins et empêcher l'attaque d'usurpation d'identité.

Après avoir déployé les nœuds capteurs, chacun découvre ses voisins et établit une clé symétrique d'un seul saut avec son voisin.

Selon le nœud voisin, qu'il ait la clé initiale ou non, on peut distinguer deux cas :

1) **Le premier cas :**

Lorsque deux voisins ayant K_{IN}, utilisent cette clé pour générer une clé partagée entre eux.

Pour l'obtention d'une clé partagée entre deux nœuds u et v, on utilise la formule suivante :

$$K_{u,v} = G(MK_{min(u,v)}, max(u,v)||N_{max(u,v)}) \qquad (1)$$

Remarque : u et v peuvent calculer $MK_{min(u,v)}$ puisque chacun connaît K_{IN} et peut générer la clé master de n'importe qu'elle nœud.

La figure 4a fait la description de différentes étapes pour l'établissement de clé *symétrique* d'un seul saut $K_{u,v}$ entre u et son voisin v avant T_{min}.

Remarque : le nœud v envoie aussi le message *Join1* au u. Après les étapes précédentes, le nœud u établit une clé *symétrique* avec chaque voisin immédiat dans la phase d'initialisation.

2)**Le deuxième cas** :

Après T_{min}, les nœuds suppriment la clé initiale transitoire K_{IN}, et deviennent des nœuds incapables de générer une autre clé master.

Alors lorsqu'un nouveau nœud u est initialisé (porte la clé K_{IN}), chaque nœud v voisin à u, ayant supprimé K_{IN} doit utiliser sa clé master pour générer une clé symétrique avec u (puisque u peut générer n'importe qu'elle clé master) :

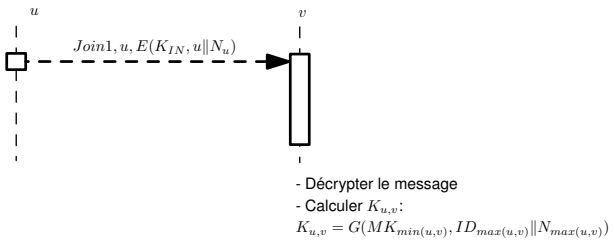
$$K_{u,v} = G(MK_v, ID_u||N_v) \qquad (2)$$

La figure 4b fait la description de différentes étapes pour l'établissement d'une clé *symétrique* d'un seul saut $K_{u,v}$ entre le nouveau nœud u et un voisin v.

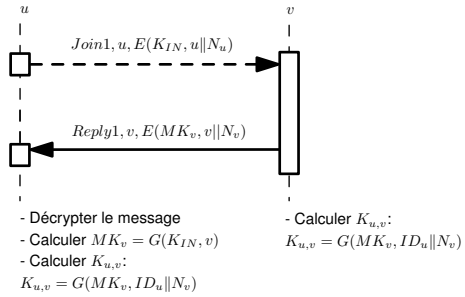
4.5.3 L'établissement de clé symétrique à deux sauts :

L'établissement des clés à deux sauts est un concept important dans **SEDAN.** L'utilisation de ce concept, permet à **SEDAN** d'éliminer l'utilisation de $\mu TESLA$ ou la référence du nœud **PUITS** pendant la phase de vérification. Par conséquent, il permet le passage à l'échelle. L'établissement des clés d'un et de deux sauts se fait de manière parallèle. Lorsqu'un nouveau nœud envoie le message *Join1*, chaque nœud voisin ayant reçu ce message devient comme un nœud relais pour les voisins à deux sauts. Selon le nœud voisin, qu'il ait la clé initiale ou non, on peut distinguer deux cas :

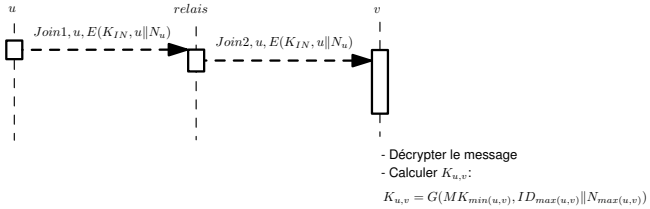
1)**Le premier cas :**



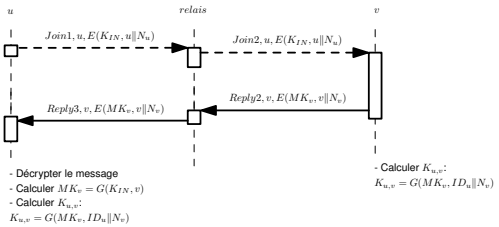
(a) Le premier cas de l'établissement des clés d'un seul saut



(b) Le deuxième cas de l'établissement des clés d'un seul saut



(c) Le premier cas de l'établissement des clés de deux sauts



(d) Le deuxième cas de l'établissement des clés de deux sauts

FIGURE 4.4: Le mécanisme EPKE de l'établissement des clés entre les voisins d'un et deux sauts.
Les lignes représentent les messages diffusés.

Comme le premier type de clés, si le nœud voisin v à deux sauts ayant la clé K_{IN} alors il peut calculer la clé *symétrique* avec un nouveau nœud u après la réception du message *Join2* à partir de nœud relais, en utilisant la formule 1.

2)**Le deuxième cas** : si le voisin à deux sauts v a déjà supprimé sa clé initiale K_{IN}, la nouvelle clé *symétrique* à deux saut doit être générée en utilisant la clé Master et le nonce de nœud v. Le nœud v envoie le message *Reply2* au nœud relais, ce dernier renvoie ce message au nouveau nœud. La nouvelle clé est établie en utilisant la formule 2.

Puisque la clé *symétrique* est générée en utilisant la clé de u ou v, si le nœud relais est un nœud compromis, il ne peut pas générer la clé *symétrique* de u et v, du fait des hypothèses, le nœud compromis (dans ce cas le nœud relais) n'a pas la clé K_{IN}.

4.6 Conclusion

Les concepteurs des protocoles dans les **RCSF**s visent à minimiser la consommation d'énergie. En effet, les RCSF sont caractérisés par des nœuds ayant chacun une unité d'énergie limitée. De plus, et après le déploiement on ne peut changer leurs unités d'énergie épuisées.

Dans les **RCSF**s, il est connu que 70% d'énergie consommée est consacrée à la com-

munication [4]. D'où, le paradigme de l'agrégation peut conserver l'énergie consommée. D'autre part, un point essentiel dans la conception des protocoles dans les **RCSF**s est celui de la sécurité. Du moment où l'agrégation élimine la redondance, elle rend la vérification de l'intégrité de données plus complexe. Dans ce chapitre nous avons proposé un protocole, appelé **SEDAN**, garantissant l'agrégation des données muni d'un mécanisme de vérification de l'intégrité. **SEDAN** gère les clés symétriques à deux sauts, ce qui permet d'éliminer la référence du nœud **PUITS**, pour la vérification de l'intégrité des données. Par conséquent, **SEDAN** minimise les messages transférés entre les noeuds capteurs et le noeuds **PUITS** en éliminant les paquets de révélation des clés, et par conséquence minimise la consommation d'énergie.

5

Analyse de sécurité et simulation

5.1 Introduction

Dans l'étude des performances des protocoles d'agrégation sécurisés, la simulation et l'analyse sont nécessaires pour mesurer l'efficacité et la robustesse. Pour cela, nous avons effectué des simulations en utilisant **TOSSIM** [39], qui est un simulateur de réseau de capteurs. **TOSSIM** compile directement du code **TinyOS** et simule la pile réseau de **TinyOS** bit par bit. Cela a l'avantage de parfaitement modéliser le comportement d'implémentation. Nous avons également utilisé **PowerTossim** [40] (une extension de **TOSSIM**) pour l'étude de l'impact de la consommation d'énergie sur ces protocoles.

En se qui concerne les métriques :

1. Nous avons utilisé la consommation d'énergie car c'est le facteur le plus important. Ce dernier paramètre a été mesuré en faisant varier le nombre de nœuds ainsi que le nombre de paquets envoyés.
2. Nous avons utilisé le facteur **MTTD** (*Le temps moyen de détection*) qui est le temps nécessaire pour détecter un nœud compromis dans le réseau. En mesurant le **MTTD** en faisant varier le nombre de nœuds, nous pouvons analyser la possibilité de passage à l'échelle de chaque protocole.

Dans la partie analyse, nous avons ciblé les problèmes suivants :

- **Le rejet aveugle :**
  Il reflète la quantité de données saines perdues à cause des nœuds malicieux en injectant des données corrompues. Nous avons étudié l'impact de ce facteur sur les protocoles en variant le nombre et la position des nœuds malicieux.
- **L'injection directe des données :**
  C'est la possibilité d'envoyer une fausse lecture par un nœud feuille sans être détectée par le nœud **PUITS**.
- **La compromission des nœuds agrégateurs :**
  C'est la possibilité qu'un nœud agrégateur falsifie les valeurs de l'agrégation.
- **L'attaque d'usurpation d'identité :**
  C'est la possibilité d'effectue des attaques sur le réseau par un nœud en injectant des paquets erronés dont l'adresse source est un autre nœud. Dans ce cas, le nœud source de ces paquets sera accusé.

5.2 Analyse des protocoles

5.2.1 Le rejet aveugle

Quand un protocole souffre de ce type de problème, il ne peut pas empêcher les données corrompues de falsifier la valeur d'agrégation finale. Notre protocole **SEDAN** vainc le rejet aveugle par l'élimination immédiate des données invalides pendant la phase d'acheminement et avant l'arrivée au nœud **PUITS**. Dans cette section, nous illustrons comment **SEDAN** et **SAWN** réagissent lors de la présence des nœuds malicieux, et nous étudions l'impact des positions et le nombre des nœuds intrus sur la quantité des données rejetées dans les branches infectées.

- **Le rejet aveugle vs position de nœud malicieux**
  Nous remarquons que **SAWN** et autres protocoles [24], [29], [30], [31] ont une quantité des données perdues de 100%. Par contre, **SEDAN** réagit de manière acceptable. Les données sont perdues, selon la profondeur de l'intrus dans l'arbre. Pour simplifier l'analyse, nous considérons l'architecture comme un *be-arbre* avec une profondeur *d*. Nous pouvons élaborer l'équation suivante pour les données perdues dans **SEDAN :**

$$DL(x) = \frac{\sum_{k=0}^{d-x} 2^k}{\sum_{k=0}^{d-1} 2^k}, 1 \leqslant x \leqslant d \qquad (1)$$

  Où : x représente la distance du nœud malicieux par rapport au nœud **PUITS**.

La Figure 5.1 illustre la variation des données perdues par rapport au nombre de sauts entre le noeud malicieux et le nœud **PUITS**, avec une profondeur de l'arbre qui est égale à 20. Néanmoins, ces résultats reflètent le fait que l'intrus n'envoie que les paquets erronés. Si l'intrus achemine les données de ses fils correctement, sauf qu'il falsifie seulement la valeur de l'agrégation, alors dans **SEDAN**, les données ne seront pas perdues, car le père du nœud malicieux peut reconstruire la valeur de l'agrégation réelle à partir des données de ses petits fils.

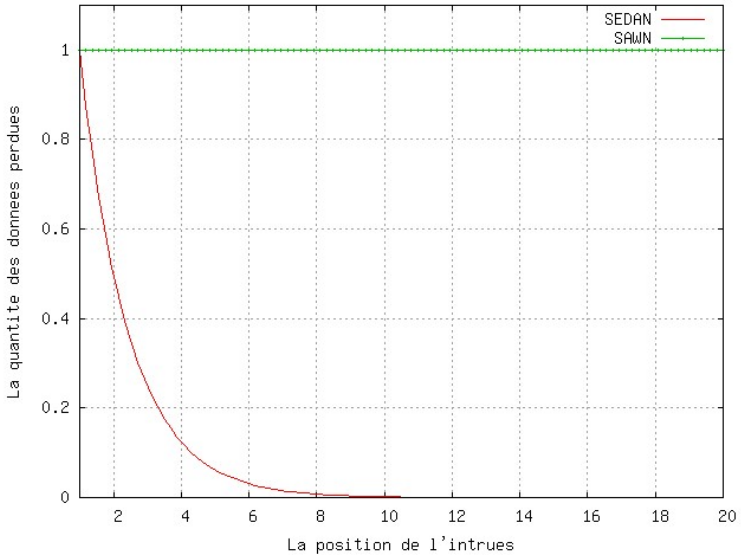


FIGURE 5.1: La quantité de données rejetées vs la position de l'intrus

– **Le rejet aveugle vs nombre de nœud malicieux**

Les pertes de données dans **SEDAN** dépendent linéairement du nombre de nœuds malicieux. Par contre, dans **SAWN** et les autres protocoles [24], [29], [30] et [31] la quantité des données perdues est toujours égale à 100% quel que soit le nombre des nœuds intrus. Pour la simplification de l'analyse, nous considérons l'architecture comme un *be-arbre* avec une profondeur d, de plus les nœuds intrus sont des nœuds feuilles seulement. Nous pouvons élaborer l'équation suivante pour les données rejetées dans **SEDAN** :

$$DL(x) = \frac{x}{2^d}, 1 \leqslant x \leqslant d \qquad (2)$$

Où :
x représente le nombre de nœuds feuilles intrus dans le réseau.
La Figure 5.2 illustre la variation des données perdues par rapport au nombre de nœuds feuilles intrus, où la profondeur de l'arbre est 10.

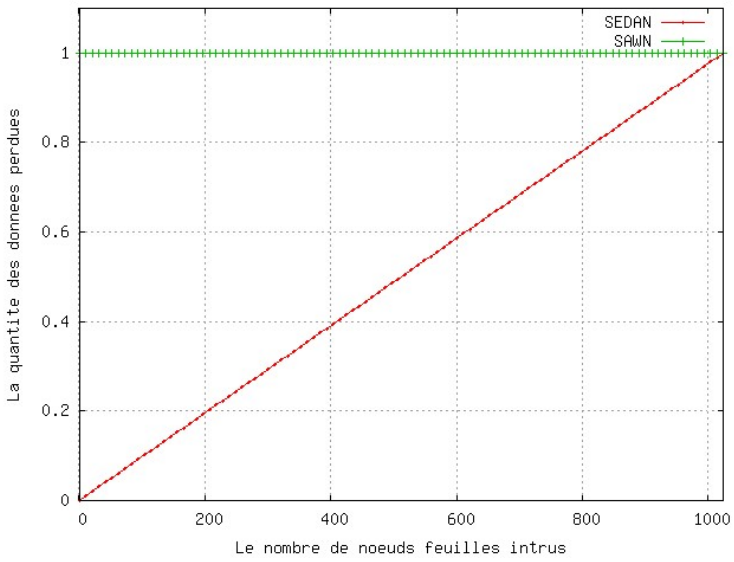


FIGURE 5.2: La quantité des données rejetées vs le nombre de nœuds feuilles intrus

5.2.2 L'injection directe des données :

L'attaque d'injection directe des données se produit lorsque l'attaquant modifie les lectures envoyées par les nœuds contrôlés directement (compromis : les nœud ayant les clés des nœuds victimes). Une telle attaque est difficile à détecter. Pour minimiser l'impact de cette attaque, les données acceptées doivent être bornées entre une valeur min et une valeur max, les valeurs min et max sont liées par le type d'application. Dans le cas où les nœuds intrus ne possèdent pas les clés des nœuds victimes, tous les protocoles peuvent détecter ces nœuds intrus.

5.2.3 La compromission des nœuds agrégateurs :

Il est très important de vérifier le comportement des nœuds agrégateurs. Un nœud agrégateur compromis peut falsifier la valeur de l'agrégation en ignorant les lectures reçues à partir de ses fils ou en modifiant leurs lectures tout simplement. Dans le premier cas, le nœud fils utilise le mécanisme de *watchdog* pour vérifier que son père a acheminé sa donnée. Dans le deuxième cas, **SEDAN** peut arrêter n'importe quelle tentative de modification au niveau du nœud parent. En vérifiant les *OHM*s des nœuds fils et tous les *THM*s des

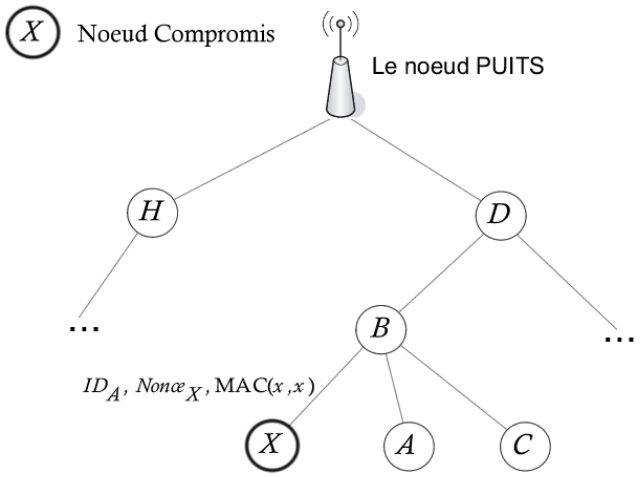


FIGURE 5.3: L'attaque d'usurpation d'identité

petits fils, un nœud peut détecter toutes les modifications possibles par son fils sur la valeur de l'agrégation.

5.2.4 L'attaque d'usurpation d'identité :

Dans, **SAWN**, lorsqu'un nœud détecte un **MAC** invalide, il doit exclure son fils et son petit fils du réseau. D'autre part, il n'y a aucun mécanisme disponible pour vérifier l'origine des paquets. Pour cela un attaquant peut effectuer une attaque d'usurpation d'identité pour supprimer des nœuds sains dans le réseau.

La figure 5.3 illustre cette attaque. Où le nœud compromis X essaie d'envoyer un message erroné à B en utilisant l'identité de A. Dans ce cas le nœud D détecte l'un des deux comme étant un noeud compromis, et par suite exclut les deux nœuds (A, B) du réseau. Dans **SEDAN**, l'utilisation des clés *symétriques* entre le nœud A et B, pour le calcul de *OHM*, permet l'authentification de l'origine des données, et rejette n'importe quel message reçu à partir d'un nœud non authentifié.

5.2.5 Le passage à l'échelle :

Le protocole **SAWN**, pendant la phase de révélation des clés, se base sur l'idée que le nœud **PUITS** soit puissant et capable de joindre tout le réseau en une seule diffusion. Cette hypothèse lie la taille du réseau à la portée du nœud **PUITS**. C'est clair que le protocole **SAWN** ne permet pas le passage à l'échelle. Par contre, la vérification dans notre contribution est totalement distribuée, et ne fait aucune référence au nœud **PUITS**, ce qui permet le passage à l'échelle.

SecureDAV, lui aussi ne permet pas le passage à l'échelle puisque la taille du réseau dépend de la portée du chef de cluster. Par contre **CMT** permet le passage à l'échelle

puisqu' il est équivalent à un processus d'agrégation simple.

5.2.6 La localisation des nœuds malicieux :

Nous avons remarqué, que les protocoles basés sur le cryptage des données de *bout-en-bout*, ne localisent pas les nœuds malicieux. Par contre, ceux qui sont basés sur le cryptage de proche-en-proche localisent ces derniers. Les protocoles **SAWN**, **SEDAN**, **SDAP** et **SecureDAV** localisent tous les nœuds compromis, ce qui n'est pas le cas dans CMT.

5.3 Les simulations

Nous avons implémenté tous les protocoles en utilisant l'environnement [34], par le biais du simulateur TOSSIM [39]. Bien entendu, **TOSSIM** est un simulateur des réseaux de capteurs sans fils, il compile l'application **TinyOS** et simule le réseau de capteurs sur l'ensemble des applications.

Pour analyser la consommation d'énergie, nous avons utilisé l'extension **PowerTossim** [40]. Cet outil donne des rapports sur l'énergie en se basant sur le model de consommation d'énergie mica2 : cpu, radio, capteur, . . .etc. Aussi, nous avons utilisé **TinySec** [41] comme une bibliothèque de cryptographie. **TinySec** contient deux algorithmes de chiffrement cryptographique : **Skipjack** et **RC5**. Dans notre simulation nous avons utilisé l'algorithme de **Skipjack** pour crypter les données ainsi que pour le calcul des **MAC**s.

Les simulations sont exécutées en utilisant différentes topologies avec un nombre moyen de voisins égal à 7. Le temps de simulation dans tous les scénarios est fixé à 100 seconds. Pour montrer l'efficacité de notre protocole **SEDAN**, nous avons mesuré deux metrics : La consommation d'énergie, et le temps moyen de détection MTTD (Mean Time To Detection)

5.3.1 La consommation d'énergie

En utilisant l'extension **PowerTossim**, nous avons mesuré la consommation de l'énergie des protocoles, **SAWN**, **SEDAN**, **CMT**, **SecureDAV** et **SDAP**. Nous avons également étudié la consommation d'énergie moyenne en variant le nombre de capteurs et le nombre de paquets de données envoyées par les nœuds feuilles. Pour faire une référence de comparaison, nous avons mesuré l'énergie consommée par un nœud dormant *(idle node)* : un nœud qui n'exécute aucun protocole, seulement sa carte réseau est allumée. La Figure 5.4 présente l'effet de l'augmentation du nombre de noeuds sur la consommation d'énergie. Pour ces simulations, nous avons fixé le nombre de paquets envoyés de chaque

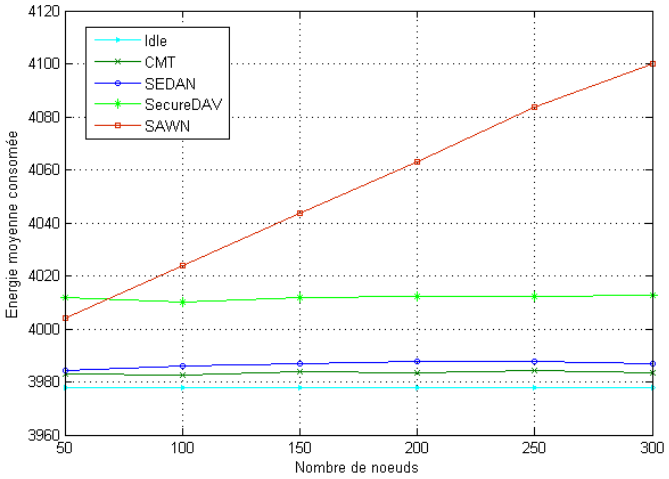


FIGURE 5.4: L'énergie consommée total vs nombre de noeuds

nœud feuille à 1 paquet par 20 seconds. Nous remarquons que la consommation d'énergie dépend du mécanisme de vérification utilisé dans chaque protocole. Par exemple **SAWN** [20] consomme beaucoup d'énergie puisque le nombre de paquets μ**TESLA** diffusés par le nœud **PUITS** est très important pour la révélation des clés utilisées dans le calcul des **MACs**. De plus, ce nombre est de l'ordre de $O(n)$ où n est le nombre de nœud dans le réseau, par conséquence ce protocole ne permet pas le passage à l'échelle. **SEDAN** dépend aussi du calcul de **MAC** pour garantir l'authentification et l'intégrité des données. Néanmoins, **SEDAN** est plus performant que **SAWN**, du fait qu'il élimine la transmission des clés. En effet, dans **SEDAN** chaque nœud partage une clé avec son père et une autre avec son grand père pour permettre la vérification instantanée sans besoin de diffusion des clés par le nœud **PUITS**. L'utilisation du ECC dans **SecureDAV**, pour signer la valeur moyenne envoyée par le chef du cluster, consomme une quantité non négligeable d'énergie, puisque il utilise un système cryptographique à clé publique. **SDAP** est un protocole basé sur la construction des cliques maximales, qui est un problème NP-complet. La construction d'une telle topologie exige un nombre important de messages, par conséquent elle consomme beaucoup d'énergie. Nous notons qu'il y a un compromis entre la consommation d'énergie d'une part et le rejet des données et la non localisation des nœuds malicieux d'autre part, comme le montre la figure 5.4. Nous remarquons aussi que **CMT** [30] (protocole basé sur le cryptage des données de *bout-en-bout*) souffre du rejet aveugle et de la non localisation des nœuds malicieux. Par contre, il ne consomme pas beaucoup d'énergie.

La Figure 5.5 montre la consommation d'énergie au niveau du *CPU* de chaque protocole, d'après le graphe, les protocoles **CMT** et **SEDAN** sont presque équivalents au model

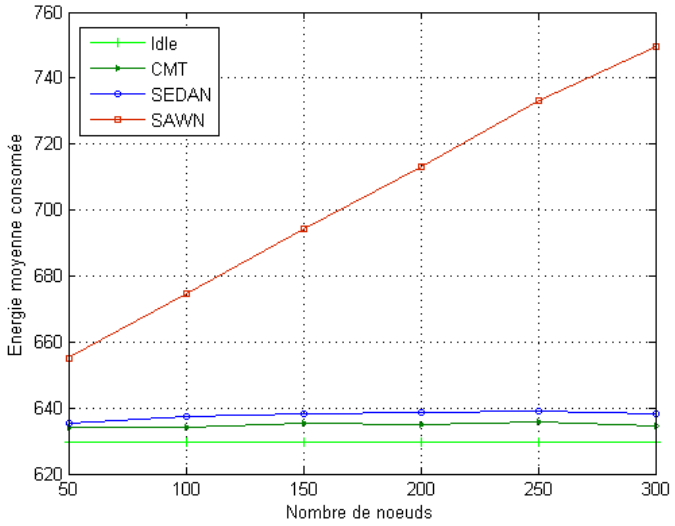


FIGURE 5.5: L'énergie consommée dans au niveau du CPU vs nombre de noeuds

Idle, puisque ce sont des protocoles qui utilisent des algorithmes symétriques, aussi le nombre de messages envoyés dans chaque protocole est équivalant au nombre de messages envoyés dans un processus d'agrégation simple. Par contre, le protocole **CMT** est légèrement meilleur par rapport à **SEDAN**, puisque les nœuds intermédiaires ne font aucun chiffrement de données. **SAWN** engendre un nombre important de messages pendant la phase de révélation des clés, où le nœud **PUITS** diffuse un nombre important de paquets dans le réseau, du fait que les nœuds reçoivent ces messages et les traitent ce qui augmente la consommation d'énergie au niveau de leurs *CPU*s.

La Figure 5.6 montre la quantité d'énergie consommée dans l'émission et la réception des messages par rapport au nombre de noeuds. Nous remarquons que la quantité d'énergie consommée dans **SAWN** est énorme, car le nœud **PUITS** dans **SAWN**, diffuse un nombre important des messages dans la phase de révélation des clés. Lorsque les nœuds capteurs reçoivent ces messages, ils consomment une quantité d'énergie importante. Par contre, dans **SEDAN** et **CMT**, le nombre de message est équivalent à un processus d'agrégation simple, pour cela ces protocoles ne consomment pas beaucoup d'énergie dans les communications.

La Figure 5.7 illustre la variation d'énergie par rapport au nombre de paquets envoyés. Nous avons fixé le nombre de nœud à 81 (une grille de 9×9). Lorsque nous analysons le comportement des protocoles en variant le nombre de paquets envoyées nous remarquons que **SEDAN** est plus scalable par rapport à **SAWN** (**SEDAN** est presque équivalent au mode *idle*).

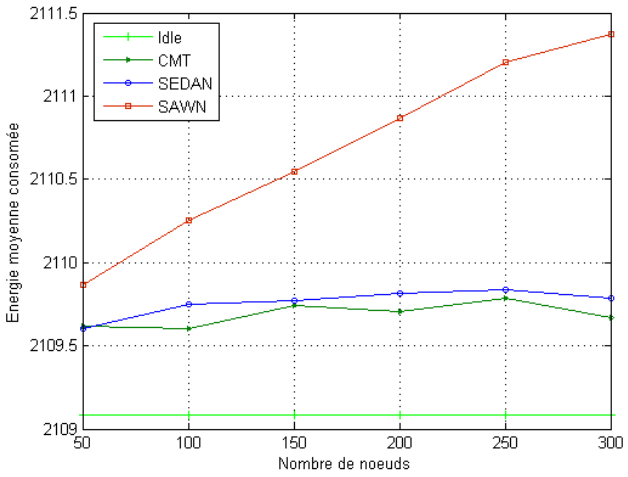


FIGURE 5.6: L'énergie consommée dans les transmissions vs nombre de noeuds

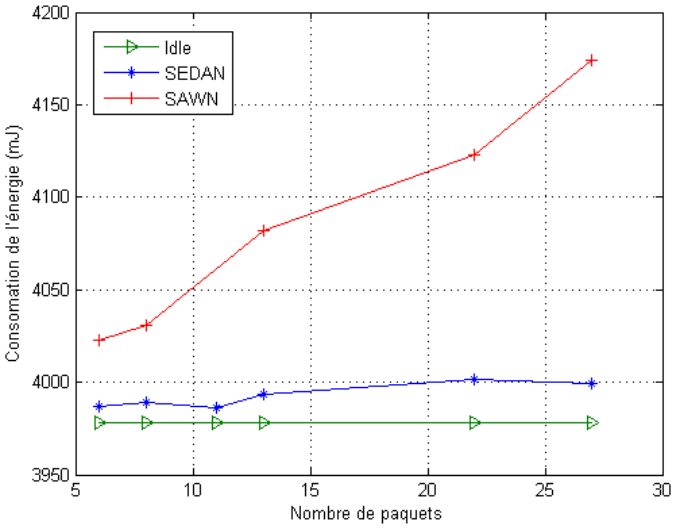


FIGURE 5.7: L'énergie consommée vs nombre de paquets envoyés

5.3.2 Le temps moyen de détection MTTD (Mean Time To Detection)

Nous entendons par le **MTTD**, le délai moyen entre l'injection d'un paquet erroné et sa détection. La Figure 5.8 illustre l'avantage de l'utilisation du mécanisme de vérification à deux sauts sans référence au noeud **PUITS**. La détection dans **SAWN** est non distribuée et la vérification est retardée jusqu'à la fin de la phase de révélation des clés de tous les noeuds. Ce retard est incrémenté lorsque le nombre de nœud est incrémenté, ainsi il a besoin d'envoyer plus de clés (voir la Figure 4.2). Contrairement au **SAWN**, dans **SEDAN** la détection est rapide, constante et proche de zéro.

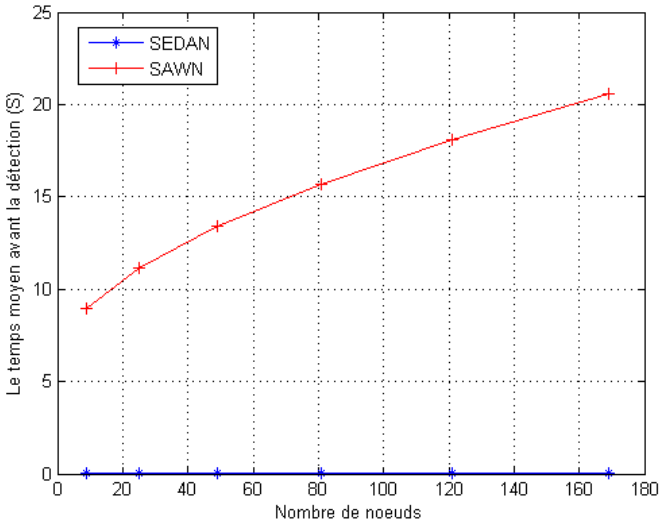


FIGURE 5.8: MTTD vs nombre de paquets

5.4 Conclusion

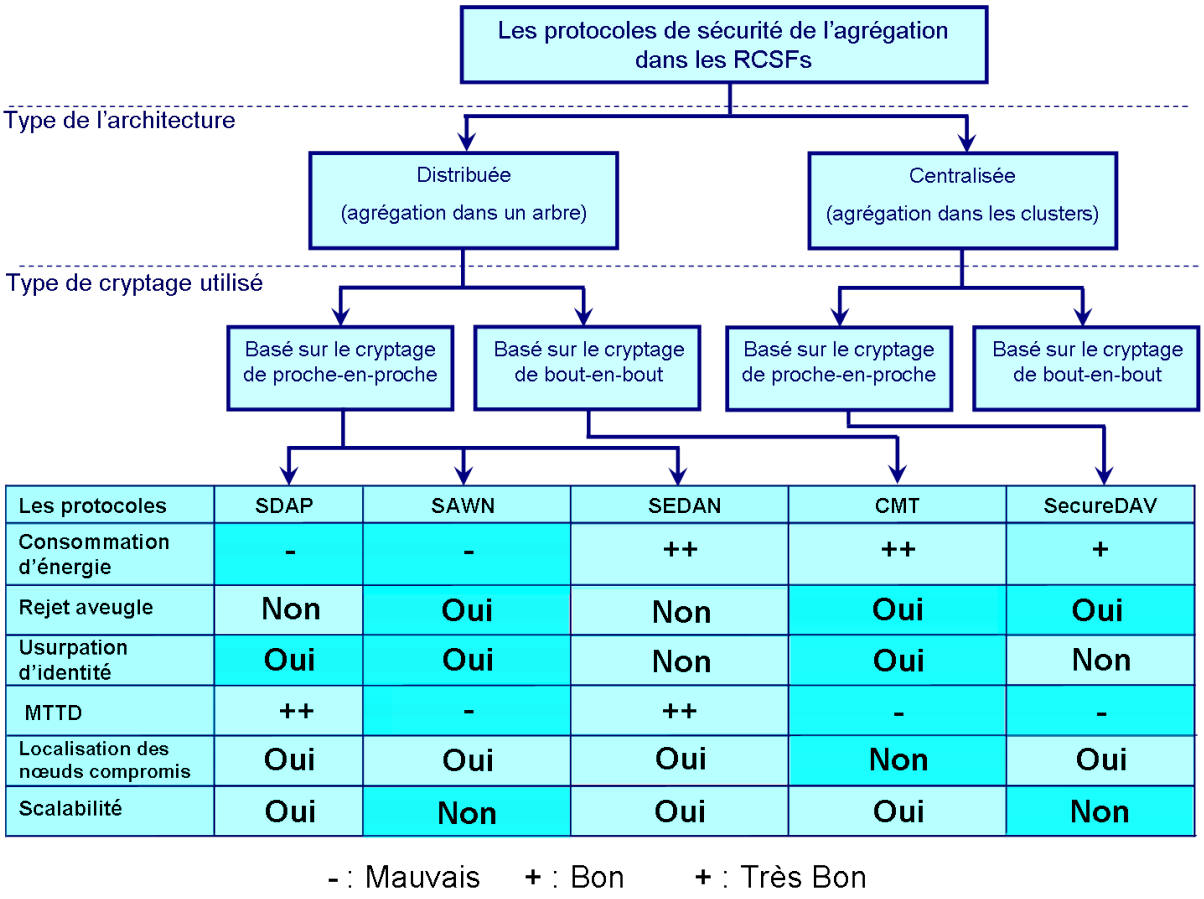
La maximisation de la durée de vie d'un réseau de capteurs requiert la progression à deux échelles complémentaires : locale et globale. Localement, chaque noeud doit optimiser sa consommation d'énergie pour allonger sa durée de vie, alors que globalement, les noeuds doivent coopérer pour optimiser la gestion des ressources en énergie.

Localement, notre protocole optimise la consommation de l'énergie en utilisant des algorithmes cryptographiques symétriques. Globalement, notre contribution optimise la consommation de l'énergie en distribuant l'agrégation et la vérification dans le réseau.

Dans ce chapitre, nous avons mené une évaluation des performances des protocoles au moyen d'une analyse et d'une simulation. Cette dernière, a été faite à l'aide du simulateur **TOSSIM**.

Les résultats présentés dans ce chapitre montrent l'efficacité de notre solution en termes de durée de vie du réseau. Cette solution contribue à l'élimination du problème du rejet aveugle, de l'attaque d'usurpation d'identité et de la non localisation des nœuds malicieux. D'après les simulations et les analyses, notre solution prouve ses performances par rapport aux protocoles existants.

Le diagramme suivant est une comparaison entre les protocoles de sécurité d'agrégation dans **RCSF** par rapport notre contribution (**SEDAN**) :



Les protocoles	SDAP	SAWN	SEDAN	CMT	SecureDAV
Consommation d'énergie	-	-	++	++	+
Rejet aveugle	Non	Oui	Non	Oui	Oui
Usurpation d'identité	Oui	Oui	Non	Oui	Non
MTTD	++	-	++	-	-
Localisation des nœuds compromis	Oui	Oui	Oui	Non	Oui
Scalabilité	Oui	Non	Oui	Oui	Non

- : Mauvais + : Bon + : Très Bon

FIGURE 5.9: Comparaison entre les protocoles de sécurité d'agrégation

Conclusion Générale

Dans ce mémoire, nous avons mis en avant les caractéristiques essentielles des réseaux de capteurs sans fil, ainsi que les différents mécanismes d'agrégation des données qui permettent de réduire considérablement l'épuisement de l'énergie, en minimisant le nombre de messages transmis dans le réseau.

En cours de transmission, des intrus tenteraient d'altérer les données agrégées ainsi que les résultats intermédiaires dans l'objectif d'inhiber ou fausser la prise de décision correcte. Afin de pallier ce type d'attaque, il est nécessaire de sécuriser l'agrégation des informations captées en assurant leur intégrité et leur authenticité.

Le problème de l'incompatibilité entre la sécurité et l'agrégation, est que cette dernière rend la tâche de sécurité plus difficile, du moment que les nœuds font des traitements sur les données des autres nœuds. Nous avons remarqué qu'il est difficile de faire cohabiter et réconcilier la sécurité et l'agrégation : d'une part, chaque nœud agrégateur doit pouvoir manipuler les données des autres nœuds, d'une autre part, les techniques d'agrégation traditionnelles comme le cryptage des données à clés publiques, ne sont pas adéquates pour les RCSF.

Nous avons étudié des protocoles de sécurité d'agrégation de données qui sont basés soit sur le cryptage de bout-en-bout ou sur le cryptage de proche-en-proche qui utilisent des mécanismes de sécurité légers. D'après les critiques de ces solutions nous avons trouvé que toutes les solutions présentent les problèmes suivants :

La centralisation de la vérification étend les temps de réponse et freine l'adaptation au facteur d'échelle.

Le rejet aveugle signifie que lors de la détection de valeurs compromises, par la modification de sa lecture ou de sa valeur d'agrégation, la valeur finale sera rejetée.

Afin de régler ces problèmes, nous avons conçus un nouveau protocole SEDAN [33] qui est basé sur le mécanisme de vérification à deux sauts. Notre solution élimine ces problèmes

en distribuant la vérification dans le réseau et en bloquant les données corrompues au niveau des nœuds malicieux.

Malheureusement, tous les protocoles y compris le notre, souffrent du problème de l'attaque « Détecteur Menteur » qui signifie la possibilité qu'un nœud compromis annonce des nœuds sains dans le réseau comme des nœuds compromis, afin de supprimer ces nœuds du réseau. Comme perspective, nous étudions une solution qui permet de résoudre ce problème. Il serait souhaitable que cette solution soit générique pour qu'elle puisse être utilisée dans les autres techniques d'agrégation de données qui souffrent du même type d'attaque.
Autre perspective serait de proposer une solution que deux noeuds consécutifs ne soient pas compromis en utilisant un nouveau type de clé.

Bibliographie

[1] http ://www.crc.ca/ site Internet de Centre de recherches sur les communications Canada.

[2] Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci "A Survey on Sensor Networks", IEEE Communications Magazine, Août 2002.

[3] : http ://www.tinyos.net/ site Internet officiel de l'OS TinyOS.

[4] : G.J. Pottie and W.J. Kaiser, "Wireless Integrated Network Sensors", Communications of the ACM, vol. 43, no. 5, pp. 51-58, May 2000.J.

[5] Imrich Chlamtac, Iacopo Carreras et Hagen Woesner "From internets to bionets : biological kinetic service oriented networks". The case study of Bionetic Sensor Networks. CREATE-NET Research Consortium, Trento, Italy 2005.

[6] Al-Karaki, J.N. and Kamal, A.E., Routing techniques in wireless sensor networks : a survey. IEEE Wireless Communications. v11 i6. 6-28.

[7] Kemal Akkaya , Mohamed Younis "A survey on routing protocols for wireless sensor networks" Ad Hoc Networks 3,2005.

[8] S. Lindsey and C. Raghavendra, "PEGASIS : Power-efficient gathering in sensor information systems," in Proceedings of IEEE Aerospace Conference, vol. 3, March 2002, pp. 1125-1130.

[9] Kai-Wei Fan, Sha Liu, and Prasun Sinha. On the potential of structure-free data aggregation in sensor networks. In IEEE INFOCOM, 2006.

[10] Chalermek Intanagonwiwat, Ramesh Govindan and Deborah Estrin. Directed diffusion : A scalable and robust communication paradigm for sensor networks. Mobile Computing and Networking, August 2000.

[11] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "An Application-Specific Protocol Architecture for Wireless Microsensor Networks," in IEEE Transactions on Wireless Communications, vol. 1, October 2002, pp. 660-670. R

[12] C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidemann. Impact of network density on data aggregation in wireless sensor networks. In Proceedings of International Conference on Distributed Computing Systems, November 2001.

[13] Yong Yao and J. E. Gehrke. The Cougar Approach to In-Network Query Processing in Sensor Networks. Sigmod Record, Volume 31, Number 3, September 2002.

[14] S. Madden, M. Franklin, J. Hellerstein, TAG : a Tiny AGgregation Service for Adhoc Sensor Networks, OSDI December 2002.

[15] J. Gehrke , Yong Yao , Query Processing in Sensor Networks, Pervasive Computing, IEEE , Volume : 3 , Issue : 1 , Jan. -March 2004 Pages :46 -55.

[16] : L. Zhao, X. Hong, and Q. Liang, "Energy-Efficient Self-Organization for Wireless Sensor Networks : A Fully Distributed Approach," in Proceedings of the 47th Annual IEEE Global Telecommunications Conference, vol. 5, November 2004, pp. 2728-2732.

[17] : S. Lindsey, C. S. Raghavendra, and K. M. Sivalingam, "Data Gathering in Sensor Networks using the Energy*Delay Metric," in Proceedings 15th International Parallel and Distributed Processing Symposium, April 2001, pp. 2001-2008.

[18] : B. J. Culpepper, L. Dung, and M. Moh, "Design and Analysis of Hybrid Indirect Transmissions (HIT) for Data Gathering in Wireless Micro Sensor Networks," in ACM SIGMOBILE Mobile Computing and Communications Review, vol. 8, January 2004, pp. 61-83.

[19] Jonathan Beaver, Mohamed A. Sharaf, Alexandros Labrinidis, Panos K. Chrysanthis, Power-Aware In-Network Query Processing for Sensor Data, MobiCom 2003.

[20] L.Hu and D. Evans, "Secure aggregation for wireless networks", Workshop on Security and Assurance in Ad Hoc Networks, January 2003.

[21] H. Ozgur Sanli, Suat Ozdemir and Hasan am "SRDA : Secure Reference-Based Data Aggregation Protocol for Wireless Sensor Networks",Proc. of IEEE VTC Fall 2004 Conference, Sept. 26-29, 2004, Los Angeles, CA, USA.

[22] L. Eschenauer and V. D. Gligor, "A key-management scheme for distributed sensor networks", Proceedings of the 9th ACM conference on Computer and communications security, Washington, DC, USA, November 18-22 2002, pp. 41-47.

[23] R. L. Rivest, M.J.B. Robshaw, R. Sidney, and Y.L. Yin, The RC6 Block Cipher, AES submission, Jun 1998. http ://theory.lcs.mit.edu/ rivest/rc6.pdf.

[24] Manik Raina et al, Secure Data Aggregation using Commitment Schemes and Quasi Commutative Functions, 2006.

[25] Mahimkar A et Rappaport, T.S, SecureDAV : A Secure Data Aggregation and Verification Protocol for Sensor Networks, 2004.

[26] Adrian Perrig, Robert Szewczyk, Victor Wen, David Culler, et Doug Tygar. SPINS : Security Protocols for Sensor Networks. Wireless Networks Journal (WINE), September 2002.

[27] R. C. Merkle, "A certified digital signature", Advances in Cryptology -Crypt0'89.

[28] K. Wua, D. Dreefa, B. Sunb, and Y. Xiao, "Secure data aggregation without persistent cryptographic operations in wireless sensor networks", Ad Hoc Networks, vol. 5, no. 1, pp. 100 - 111, 2006.

[29] Josep Domingo-Ferrer. "A provably secure additive and multiplicative privacy homomorphism". In ISC '02 : Proceedings of the 5th International Conference on Information Security, 2002.

[30] Claude Castelluccia, Einar Mykletun, and Gene Tsudik. "Effcient aggregation of encrypted data in wireless sensor networks". In MobiQuitous, pages 109-117. IEEE Computer Society, 2005.

[31] Einar Mykletun, Joao Girao, and Dirk Westhoff. "Public key based cryptoschemes for data concealment in wireless sensor networks". In IEEE International Conference on Communications. ICC2006, 2006.

[32] Joao Girao, Dirk Westhoff, and Markus Schneider. Cda : "Concealed data aggregation for reverse multicast traffic in wireless sensor networks". In IEEE International Conference on Communications, 2005.

[33] M.Bagaa, N. Lasla, A. Ouadjaout and Y.Challal ; "SEDAN : Secure and Efficient protocol for Data Aggregation in wireless sensor Networks", 32nd IEEE Conference on Local Computer Networks (LCN 2007) pp. 1053-1060, Worksohop on Netwok Security.

[34] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister, "System architecture directions for networked sensors," in Proceedingsof Architectural Support for Programming Languages and Operating Systems, 2000, pp. 93 - 104.

[35] S. Zhu, S. Setia, and S. Jajodia, "LEAP : Efficient security mechanisms for large-scale distributed sensor networks," in Proceedings of ACM CCS, 2003.

[36] J. Deng, C. Hartung, R. Han, and S. Mishra, "A practical study of transitory master key establishment for wireless sensor networks," in Proceedingsof the First International Conference on Security and Privacy for Emerging Areas in Communications Networks (SECURECOMM 05), 2005, pp. 289 - 302.

[37] : W. Du, J. D. Eng, Y. S. Han, and V. Arshney, "A pairwise key predistribution scheme for wireless sensor networks," in Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS03), 2003, pp. 42 - 51.

[38] Y.WANG, G.ATTEBURY and B.RAMAMURTHY "A SURVEY OF SECURITY ISSUES IN WIRELESS SENSOR NETWORKS", in IEEE Communication Survey Tutorials, 2006.

[39] L. Philip, L. Nelson, W. Matt, and C. David, "Tossim : Accurate and scalable simulation of entire tinyos applications," in Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys2003). 2003.

[40] V. Shnayder, M. Hempstead, B. Chen, G. W. Allen, and M. Welsh, "Simulating the power consumption of large-scale sensor network applications," in Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems. New York, NY, USA : ACM Press, 2004, pp. 188–200.

[41] C. Karlof, N. Sastry, and D. Wagner, "Tinysec : A link layer security architecture for wireless sensor networks," in Second ACM Conference on Embedded Networked Sensor Systems (SensSys 2004), November 2004.