SPIM

Thèse de Doctorat

école doctorale **sciences pour l'ingénieur et microtechniques**

UFC UNIVERSITÉ DE FRANCHE-COMTÉ

Cryptographie sur les courbes elliptiques et tolérance aux pannes dans les réseaux de capteurs

YANBO SHOU

SPIM

Thèse de Doctorat

école doctorale **sciences pour l'ingénieur et microtechniques**

UFC UNIVERSITÉ DE FRANCHE-COMTÉ

N° | X | X | X |

THÈSE présentée par

YANBO SHOU

pour obtenir le

Grade de Docteur de
l'Université de Franche-Comté

Spécialité : **Informatique**

Cryptographie sur les courbes elliptiques et tolérance aux pannes dans les réseaux de capteurs

Soutenue publiquement le 10/09/2014 devant le Jury composé de :

HERVÉ GUYENNET	Directeur de thèse	Professeur à l'Université de Franche-Comté
J.-C. LAPAYRE	Examinateur	Professeur à l'Université de Franche-Comté
MARC BUI	Rapporteur	Professeur à l'EPHE - Sorbonne, Paris
ABDELHAMID MELLOUK	Rapporteur	Professeur à l'Université Paris-Est Créteil

REMERCIEMENTS

La réalisation d'une thèse n'est pas le résultat du travail d'un seul homme, mais de son interaction avec tous ceux gravitant autour de lui pour l'aider théoriquement, techniquement et humainement. Ainsi je tiens à remercier tous ceux qui m'ont de prêt ou de loin apporter leur soutien :

- en premier lieu Hervé Guyennet, qui par son écoute et ses précieux conseils m'a permis de mener à terme cette aventure. Il aura été au cours de ces 3 ans bien plus qu'un simple directeur de thèse et je le remercie de m'avoir accordé sa confiance et de m'avoir permis de développer mes idées.
- Marc Bui et Abdelhamid Mellouk pour avoir accepté de donner de leurs temps libre pour rapporter cette thèse et apporter des critiques pertinentes et constructives.
- Jean-Christophe Lapayre pour l'intérêt qu'il a suscité de mon travail et pour avoir accepté de participer à mon jury.
- Les membres du Femto-ST/DISC, maîtres de conférences et professeurs qui m'ont accueilli et grâce à qui se rendre au laboratoire n'était pas un fardeau.
- Mes collègues de l'équipe CARTOON : Youssou Faye, Mathias Coqblin, Julien Bernard, Adel Elgaber, Violetta Felea pour leur sympathie et les discussions de recherche constructives.
- Tous les doctorants et anciens doctorants pour le temps passé ensemble et l'esprit de corps qu'ils ont su maintenir.
- Mes parents qui ont été une autre source de soutien et d'encouragement et sans qui je n'aurais jamais pu prétendre atteindre un si haut niveau universitaire. Ils m'ont toujours soutenu dans mes choix et je les en remercie infiniment.
- Ma famille et mes amis qui ont toujours été présents pour moi.
- Tous ceux que j'ai oublié, qui par des mots ou des actes ont aidé à la construction de cette thèse.

SOMMAIRE

TABLE DES FIGURES

LISTE DES TABLES

1

INTRODUCTION GÉNÉRALE

Grâce au développement très rapide du domaine de la micro-électronique, ces dernières années ont été marquées par un grand progrès des techniques des réseaux de capteurs sans fil. Ces petits dispositifs compacts, appelés aussi des nœuds intelligents, peuvent être programmés et déployés dans une zone d'intérêt pour mesurer des grandeurs physiques. Ils sont équipés des technologies de communication sans fil qui leur permettent de communiquer entre eux pour former automatiquement des réseaux et envoyer des données collectées.

Le nœud intelligent, qui fait partie du domaine de l'électronique, est en effet un nouveau venu du monde informatique. La nécessité d'une couche logicielle a fait intervenir des informaticiens qui essaient de mettre en œuvre tous les moyens possibles pour perfectionner son fonctionnement et diversifier ses fonctionnalités.

Aujourd'hui, suite à l'essor des systèmes embarqués et l'arrivée de l'Internet des objets, ce genre de dispositifs sont de plus en plus présents partout dans notre vie quotidienne. Cependant, le déploiement et l'interconnexion massifs des nœuds ont aussi relevés de grands défis technologiques qui ont intéressé de nombreux chercheurs. Ainsi, de nouvelles thématiques de recherche ont été créées par la communauté scientifique pour répondre aux exigences de plusieurs domaines d'application, comme la surveillance des catastrophes naturelles, la protection de l'environnement et le suivi des traitements des patients [2, 112].

Comme les réseaux traditionnels, les réseaux de capteurs subissent aussi de nombreuses attaques, qui consistent à espionner et perturber les communications entre les nœuds. Depuis des siècles, la sécurité d'informations est toujours un des sujets les plus discutés, et l'évolution de la société humaine a aussi renforcé son importance, notamment dans les domaines militaires et industriels. C'est aussi la raison pour laquelle d'importants moyens financiers sont mis en place par les états ou les grandes entreprises pour s'assurer de la confidentialité de leurs informations ou pour contourner ces sécurités et récupérer des informations cruciales.

Dans le monde informatique, l'augmentation impressionnante de la puissance de calcul des processeurs permet de déchiffrer un message dans un temps de plus en plus court. De ce fait, afin d'augmenter la sécurité d'un système, la solution la plus adaptée est d'augmenter la taille des clés de chiffrement pour augmenter le temps nécessaire à déchiffrer un message [65].

Dans les réseaux numériques, la cryptographie est toujours considérée comme une des solutions dominantes pour assurer la confidentialité des informations. La nature de cette

solution permet de rendre un message illisible en utilisant un ensemble de méthodes mathématiques, qui sont conçues pour réaliser une telle transformation en leur fournissant une clé de chiffrement. Hormis le chiffrement, elle utilise des méthodes complémentaires, qui nous offrent une protection plus sûre en assurant l'authenticité et l'intégrité.

L'utilisation de la cryptographie implique souvent des calculs compliqués et intensifs, qui ne posent pas de problèmes sérieux pour les systèmes sans contrainte de ressources. La plupart de cryptosystèmes peuvent montrer une performance satisfaisante sur les processeurs modernes, qui possèdent une puissance de calcul suffisamment élevée pour gérer ce genre de calculs. Cependant l'application de la cryptographie dans les réseaux de capteurs reste encore un challenge à relever.

La plupart des nœuds intelligents utilisent des micro-contrôleurs comme unité de traitement, qui est aussi considérée comme le composant le plus important. Un micro-contrôleur est chargé de la coordination de toutes les opérations qu'un nœud peut effectuer, c'est un circuit intégré programmable qui rassemble tous les éléments essentiels d'un ordinateur, comme le processeur, la mémoire et les interfaces entrée/sortie. Il possède souvent une puissance de calcul et une mémoire très limitée, son processeur est censé exécuter de petits programmes qui effectuent des opérations simple. Sans optimisation extrême et technique spéciale, un micro-contrôleur n'est pas capable de gérer des calculs mathématiques très compliqués.

Il existe 2 types de cryptographie principales, symétrique et asymétrique. La cryptographie symétrique offre une performance de calcul plus intéressante sans utiliser une clé extrêmement longue. Cependant, comme on partage la même clé pour le chiffrement et le déchiffrement, au sein d'un réseau de capteurs contenant un grand nombre de nœuds, la mise en place d'une distribution sûre des clés devient donc un problème urgent et sérieux.

Contrairement à la cryptographie symétrique, la cryptographie asymétrique offre des protocoles sophistiqués pour la génération de clés, et elle nous permet aussi de signer des messages. Cependant, l'application de cette solution nécessite l'utilisation des clés beaucoup plus longues et des calculs plus complexes. Aujourd'hui, RSA (Ron - Shamir - Adleman) est toujours le cryptosystème asymétrique le plus utilisé, mais pour pouvoir avoir une sécurité assez robuste, il faut utiliser une clé dont la longueur est comprise entre 1024 et 2048. Par rapport à RSA, ECC (Cryptographie sur les Courbes Elliptiques) est un autre cryptosystème asymétrique qui a récemment attiré l'attention des chercheurs, car elle peut offrir la même robustesse que RSA avec une clé beaucoup plus courte, et il existe aussi des méthodes mathématiques qui nous permettent d'améliorer sa performance. Certains chercheurs pensent que ECC deviendra le remplaçant de RSA, notamment dans le domaine des systèmes embarqués.

C'est pourquoi pendant nos travaux de cherche, nous avons étudié la possibilité d'appliquer l'ECC dans les réseaux de capteurs d'une manière efficace, car les opérations sur les courbes elliptiques sont encore très compliquées pour les micro-contrôleurs, notamment la multiplication de point, appelée aussi la multiplication scalaire, qui est considérée comme l'opération la plus coûteuse sur les courbes.

Dans cette thèse, nous cherchons à accélérer le calcul des multiplications scalaires en utilisant la technique de parallélisation qui consiste à découper le calcul en plusieurs tâches indépendantes qui peuvent être traitées simultanément par des nœuds différents. L'intérêt principal d'un déploiement massif des nœuds est d'avoir beaucoup de nœuds

disponibles qui peuvent coopérer étroitement pour réaliser un objectif commun. De plus, comme les nœuds sont des dispositifs fragiles qui risquent de tomber en panne pendant le traitement des tâches, dans cette thèse, nous proposons également une technique de tolérance aux pannes pour rendre la procédure du calcul parallèle plus stable.

PLAN DE LA THÈSE

Cette thèse est composées de 5 chapitres :

Le premier chapitre introduit le terme de réseaux de capteurs sans fil en présentant l'architecture des nœuds et les domaines d'application. Ensuite nous présentons les spécificités inhérentes aux réseaux de capteurs, comme la consommation d'énergie, la puissance de calcul etc. La partie la plus importante dans ce chapitre est une taxonomie des attaques qui peuvent sérieusement menacer la sécurité d'informations dans ce genre de réseau. Pour terminer, nous donnons une liste de solutions qui sont fréquemment utilisées dans la littérature pour protéger les réseaux.

Dans le chapitre 3, nous présentons un état de l'art de la cryptographie sur les courbes elliptiques. Nous commençons par des concepts mathématiques qui sont indispensables pour comprendre le fonctionnement d'ECC. Ensuite nous donnons la définition de courbe elliptique, les opérations élémentaires, ainsi que le problème du logarithme discret sur lequel se base le cryptosystème. Dans les sections suivantes, nous présentons un ensemble de méthodes mathématiques qui nous permettent d'accélérer les calculs sur les courbes elliptiques, et les protocoles cryptographiques principaux qui sont basés sur ECC. La dernière partie du chapitre est une comparaison entre ECC et RSA, l'autre cryptosystème asymétrique qui est largement utilisé en pratique.

Afin d'accélérer le calcul des multiplications scalaires sur les courbes elliptiques, dans le chapitre 4, nous proposons d'appliquer le parallélisme au sein des réseaux de capteurs. Avant de donner notre solution, nous présentons d'abord les facteurs qu'il faut prendre en considération lors du déploiement d'un système de calcul parallèle, ensuite les spécificités des réseaux de capteurs dont il faut tenir compte quand on essaie d'implanter cette architecture dans ce genre de réseau. Puis nous présentons notre technique de parallélisation avec l'ensemble d'algorithmes et de protocoles que nous avons utilisé pour l'implémentation. Enfin, nous validons notre solution avec un test de performance qui montre que la parallélisation peut donner une accélération de calcul importante malgré une augmentation de consommation d'énergie.

Le calcul parallèle est une procédure complexe qui risque d'être interrompu par des pannes éventuelles. En outre, les nœuds sont très fragiles face aux environnements difficiles dans lesquels ils sont déployés. C'est pourquoi nous nous intéressons dans le chapitre 5 au problème de tolérance aux pannes. Nous commençons par l'identification des objectifs principaux des mécanismes de tolérance aux pannes, ensuite nous présentons les méthodes de diagnostic qui existent dans la littérature, et puis nous nous rendons compte que la solution la plus efficace et facile à mettre en place est la redondance. Nous étudions donc les différentes techniques qui sont basées sur la redondance que nous pouvons mettre en place pour restaurer les fonctionnalités des réseaux en cas de pannes.

Nous avons identifié 3 types de panne qui peuvent éventuellement survenir pendant le

calcul parallèle des multiplications scalaires : résultat perdu, résultat erroné et nœud maître défectueux. Pour que le réseau puisse détecter les pannes et restaurer le calcul parallèle, nous présentons dans le chapitre 6 la solution de tolérance aux pannes que nous avons proposée. Afin de tester ses performances, nous avons développé un simulateur qui peut exécuter la procédure de calcul parallèle en prenant en compte les mécanismes de tolérance aux pannes proposés. Les résultats ont montré que notre solution peut restaurer le calcul parallèle en cas des pannes sans avoir un impact négatif important.

Nous concluons cette thèse dans le dernier chapitre en rappelant les avantages et les inconvénients des solutions proposées, ensuite nous donnons également des perspectives des travaux de recherche en vue d'améliorer leurs performances dans les réseaux de capteurs.

2

RÉSEAUX DE CAPTEURS ET PROBLÈMES DE SÉCURITÉ

Sommaire

2.1/ LES RÉSEAUX DE CAPTEURS SANS FIL

2.1.1/ DÉFINITION D'UN CAPTEUR

Un capteur est un petit dispositif électronique qui peut transformer une mesure physique en une mesure électronique, qui sera ensuite traduite en données binaires compréhensibles pour des systèmes informatiques. Aujourd'hui il existe différents capteurs qui sont conçus pour mesurer des valeurs diverses, par exemple la température, l'humidité, la luminosité, les mouvements, la pression etc.

Généralement, les premiers capteurs ne pouvaient lire qu'un seul type de mesure, alors que maintenant plusieurs capteurs sont installés sur un seul *nœud* pour lire des valeurs diverses.

Grâce aux progrès récents du domaine des systèmes micro-électroniques, les nouvelles technologies de la communication sans fil et de l'électronique numérique ont été intégrées dans les nœuds modernes. Hormis la fonctionnalité de base, c'est-à-dire l'acquisition des données ambiantes, les nouveaux nœuds détiennent encore des modules de transmission, d'alimentation, de stockage et de traitement de données, qui leur permettent d'envoyer, stocker et traiter les données collectées.

Un autre terme qui est largement utilisé dans la littérature est *nœud intelligent*, qui désigne particulièrement les nœuds programmables. Le développement des technologies des systèmes embarqués a donné la possibilité d'utiliser le même modèle de nœud pour des applications différentes. Les nœuds du même modèle peuvent être déployés pour assurer des missions complètement différentes, et il suffit de changer le programme installé. Par ailleurs, l'apparition des systèmes d'exploitation pour les nœuds a encore facilité la création des applications en divisant la procédure en 2 parties indépendantes : le développement des programmes et la conception de la plate-forme matérielle.

2.1.2/ ARCHITECTURE DES CAPTEURS

L'architecture des capteurs a beaucoup évolué depuis les années 90. Les capteurs de la première génération ont une architecture très simple, qui ne possède qu'une unité de capture et une unité d'alimentation. Comme les détecteurs de fumée qui sont encore beaucoup utilisés dans les immeubles. Ils sont souvent alimentés par des piles, et la seule fonctionnalité est de déclencher un alarme sonore en présence de fumée.

Sur les capteurs un peu plus évolués, on a commencé à intégrer de nouvelles technologies, comme la communication sans fil. Par exemple ; les bouchons de valve de pneu, qui sont utilisés pour mesurer les pressions des pneus et sont aussi alimentés par des piles. De plus, ils peuvent envoyer les valeurs au récepteur qui est installé à l'intérieur de la voiture.

L'architecture des nœuds actuels est devenue beaucoup plus complexe après l'ajout des fonctionnalités complémentaires, comme le traitement et le stockage de données. La figure 2.1 illustre une architecture générale des nœuds intelligents actuels. Nous pouvons constater qu'elle est constituée de 4 unités principales.

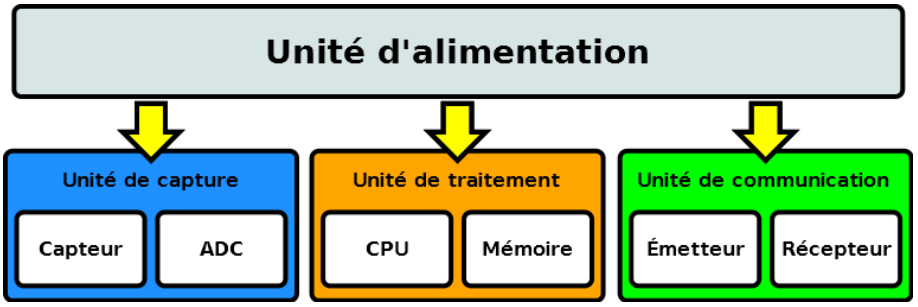


FIGURE 2.1 – Architecture d'un nœud sans fil actuel

– Unité de traitement : C'est l'unité la plus importante d'un nœud intelligent, comme les autres systèmes embarqués, on utilise généralement un micro-contrôleur qui rassemble tous les éléments essentiels d'un ordinateur, le processeur, la mémoire et les interfaces entrées-sorties.
– Unité de capture : C'est l'ensemble de composants matériels qui sont utilisés pour mesurer des grandeurs physiques et générer des signaux analogiques, ils sont couplés d'un ADC [1] qui transforme les signaux analogiques en données numériques pour l'unité de traitement.
– Unité de communication : Elle désigne généralement l'antenne radio du nœud, qui est chargée de l'émission et de la réception des signaux radiofréquence. Dans la littérature elle s'appelle aussi *transceiver*, qui est en effet une contraction des mots anglophones transmitter et receiver. À l'aide de cette unité, on peut construire des réseaux de capteurs en reliant les nœuds qui sont déployés dans la même zone.
– Unité d'alimentation : Une unité cruciale pour toute l'architecture, car c'est elle qui fournit l'énergie à toutes les autres unités. elle correspond souvent à une pile et une batterie qui s'épuise graduellement au fil du temps. À cause de la limitation des ressources des nœuds, la consommation d'énergie est devenu un facteur critique dans quasiment toutes les applications de capteurs. La réalisation récente d'unité d'alimentation tend à résoudre le problème en utilisant des panneaux solaires [61, 106].

Sur les nœuds d'usages particuliers, on peut encore ajouter d'autres unités, telle qu'une unité de localisation, comme le GPS, ou une unité de mobilité pour des nœuds qui ont besoin de se déplacer.

2.1.3/ RÉSEAUX DE CAPTEURS

Un réseau de capteurs est constitué généralement d'un grand nombre de nœuds présentés dans la section précédente. Ils sont déployés aléatoirement dans une zone pour surveiller et étudier des phénomènes divers. Les nœuds communiquent entre eux via des connexions sans fil pour le partage et le traitement coopératif des données collectées.

Après le déploiement, les nœuds lancent une procédure d'auto-configuration qui leur permet de construire des chemins de routage, sans lesquels on ne peut pas envoyer des données à un nœud spécifique. À cause d'un manque d'infrastructure de réseau et d'une limitation des ressources des nœuds, la transmission de données est souvent réalisée avec un mode de communication particulier qui s'appelle multi-saut. Les données

1. Analog-to-Digital Converter

passent d'un nœud à un autre, pas à pas, jusqu'à la destination.

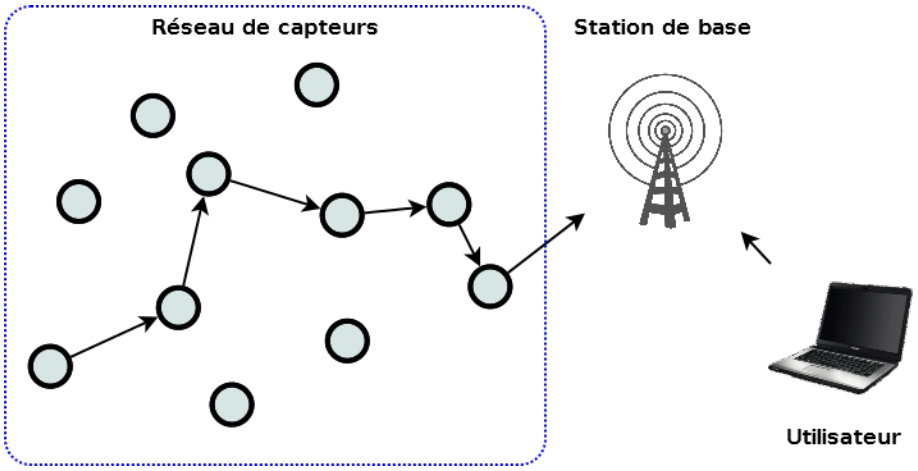


FIGURE 2.2 – Exemple de réseaux de capteurs sans fil

Dans la figure 2.2 nous avons un exemple de réseaux de capteurs. Les nœuds sont déployés aléatoirement dans une zone d'intérêt et la station de base se situe à distance. En faisant des multi-sauts, les données remontent progressivement jusqu'à la station de base, qui les met à disposition aux utilisateurs finaux.

Il existe principalement 2 types de réseaux de capteurs :
- Les réseaux actifs, dans lesquels les nœuds envoient *périodiquement* des données collectées à la station de base. Ce genre de fonctionnement est préférable pour les applications de surveillance à long terme, car il nous permet d'avoir un accès à toutes les historiques des données collectées et de mieux étudier les phénomènes surveillés.
- Les réseaux passifs, qui n'envoient aucune donnée tant qu'aucun événement important n'est détecté. On peut trouver ce genre d'application dans des systèmes de pré-alarme qui doivent signaler immédiatement l'arrivée éventuelle d'incidents importants, comme la détection des catastrophes naturelles.

2.1.4/ DOMAINES D'APPLICATION

Idéalement les réseaux de capteurs sont censés être construits en utilisant des nœuds à faible coût qui sont déployés en masse. Aujourd'hui ils ne sont pas encore largement utilisés dans notre vie quotidienne, le coût de fabrication des nœuds est toujours très élevé, certaines techniques sont aussi loin d'être matures et fiables. Cependant on peut déjà trouver beaucoup d'applications, dont certaines sont encore expérimentales, dans des domaines divers.

2.1.4.1/ APPLICATIONS MILITAIRES

Les caractéristiques attendues des réseaux de capteurs, comme le faible coût, l'auto-organisation, la sécurité et la tolérance aux pannes, ont donné naissance à son utilisation dans les domaines militaires, c'est aussi le domaine dans lequel on trouve souvent les technologies les plus avancées.

Comme le projet DSN (Distributed Sensor Network) du DARPA (Defense Advanced Research Project Agency) [16], qui était un des premiers projets dans les année 80 à avoir utilisé les réseaux de capteurs. Les nœuds peuvent être déployés rapidement dans un champ de bataille pour assurer, d'une manière discrète, des missions confidentielles, comme la collecte de renseignements, la surveillance des forces ennemies et la détection des attaques biochimiques.

2.1.4.2/ APPLICATIONS ENVIRONNEMENTALES

L'industrialisation et l'expansion humaine ont causé des problèmes environnementaux sévères, comme la pollution d'air et d'eau, la disparition des forêts etc. Notamment dans les pays en développement dont les gouvernements font rarement attention aux soucis écologiques. L'utilisation des réseaux de capteurs pour la protection de l'environnement a été beaucoup encouragée par la communauté scientifique. Un déploiement approprié des nœuds nous permet non seulement de collecter précisément des paramètres environnementaux, mais aussi de faciliter la surveillance de la zone d'intérêt et d'économiser l'investissement à long terme. Les nœuds peuvent aussi être déployés dans des zones difficilement accessibles et hostiles, une fois mis en place, ils peuvent fonctionner d'une manière autonome.

FIGURE 2.3 – Capteurs déployés pour la protection d'environnement

Dans la figure 2.3, nous avons des nœuds qui sont déployés pour surveiller des forêts et l'air. En pratique, les nœuds qui sont utilisés dans ce genre d'applications sont souvent équipés d'une protection physique solide, et afin d'avoir une durée de vie plus longue et rentable, ils sont généralement bien alimentés avec soit un accès aux réseaux électriques, soit des panneaux solaires. A cause d'une augmentation du coût d'installation, on ne peut pas effectuer un véritable déploiement à grande échelle.

2.1.4.3/ APPLICATIONS MÉDICALES

Dans le domaine de la médecine, les nœuds sont souvent utilisés pour assurer une surveillance permanente d'un patient ou d'une personne âgée. Ils peuvent être soit attachés au bras du patient, soit implantés directement sous la peau pour mesurer des paramètres physiologiques, comme la tension artérielle et le battement de cœur (voir figure 2.4).

Les nœuds peuvent communiquer avec les autres nœuds fixes installés dans l'immeuble pour récupérer les données collectées, qui sont ensuite stockées dans une base de don-

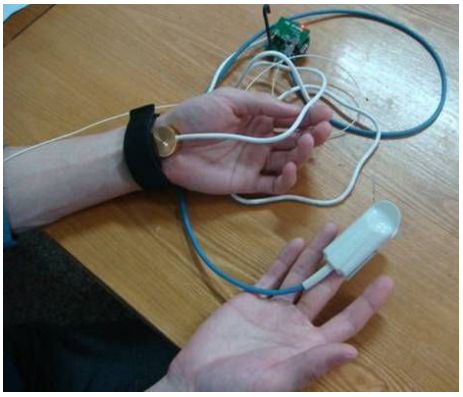


FIGURE 2.4 – Mesure de la tension artérielle et le battement de cœur avec des capteurs

nées, avec laquelle on peut avoir un suivi du patient [41]. En outre, dans certaines applications, le patient porte une unité de transmission de données (DTU [2]) ou un téléphone mobile qui récupère les données mesurées et les renvoie au serveur à distance via les réseaux mobiles.

2.1.4.4/ DOMOTIQUE

L'idée est d'intégrer des capteurs dans les appareils électroménagers, tels que le climatiseur, la chauffage, les volets électriques, les thermomètres numériques etc. Ces systèmes embarqués peuvent communiquer entre eux via des connexions sans fil, exactement comme les réseaux de capteurs. Une centrale de contrôle est mise en place pour envoyer des requêtes aux appareils et recevoir des informations. La centrale dispose encore d'un accès internet qui permet aux utilisateurs de contrôler des appareils à distance et de consulter les informations concernant son domicile [78].

Maintenant on utilise le terme *Smart Home* pour désigner l'ensemble de technologies qui permettent d'automatiser et de faciliter la gestion des immeubles avec les systèmes embarqués [99, 33, 88]. L'objectif est d'apporter aux utilisateurs un confort plus élevé en assurant une vie plus simple et écologique [59, 88]. Par exemple pendant la journée, si les capteurs détectent une forte luminosité, ils vont démarrer le moteur pour ouvrir le volet. Si les capteurs de lumière et de température trouvent que le soleil est suffisamment fort pour chauffer la chambre, ils vont couper la chauffage pour économiser la consommation d'énergie. Suite à l'essor des smartphones, les utilisateurs peuvent envoyer des requêtes depuis leurs téléphones pour contrôler des appareils chez eux.

2.2/ SPÉCIFICITÉS DES RÉSEAUX DE CAPTEURS

Un réseau de capteurs est une sorte de réseau ad-hoc, les nœuds déployés communiquent entre elles via des communications sans fil pour former un réseau et créer des chemins de routage. Le réseau ne détient aucune infrastructure, chaque nœud communique directement avec ses voisins, et les données sont envoyées en utilisant la mode de

2. Data Transfer Unit

communication multi-sauts, elles sont transmises d'un nœud à l'autre jusqu'à ce qu'elles arrivent à destination.

Hormis les caractéristiques basiques des réseaux ad-hoc, les réseaux de capteurs possèdent encore d'autres spécificités qu'il faut prendre en compte lors de la conception des applications.

2.2.1/ ÉNERGIE

Nous commençons par la consommation d'énergie dans les réseaux de capteurs, car elle est considérée comme un des facteurs les plus cruciaux. Idéalement les réseaux de capteurs sont censés être constitués des nœuds à faible coût, qui peuvent être déployés facilement en masse sans se soucier de l'organisation et de la configuration. Une fois réveillés, les nœuds peuvent fonctionner d'une manière autonome sans intervention humaine. Comme il y a des chances que les réseaux soient déployés dans des zones hostiles, il faut qu'ils permettent un minimum de tolérance aux pannes pour endurer les environnements difficiles.

Pour l'instant, la seule source d'énergie des nœuds est la pile dont la durée de vie peut varier entre des heures et des jours en fonction de l'intensité des opérations à effectuer. C'est aussi le seul mode d'alimentation qui donne la possibilité d'effectuer un déploiement aléatoire. Cependant un tel déploiement rend la maintenance manuelle des nœuds très difficile. Il est très coûteux et quasiment impossible de localiser un nœud dont la pile est épuisée.

Pour une raison de coût de fabrication et de maintenance, au lieu de chercher à remplacer des piles épuisées ou installer des piles de grande capacité, on préfère plutôt minimiser la consommation d'énergie pendant le fonctionnement des réseaux. Sachant que la plupart de l'énergie est consommée pendant des communications sans fil et que la consommation des autres composants est presque négligeable par rapport à celle de l'antenne radio, quand on conçoit une application, on essaie toujours de limiter le nombre de communications et d'éteindre la radio quand ce n'est pas nécessaire.

2.2.2/ PUISSANCE DE CALCUL

Toujours pour une raison de coût, on choisit souvent des micro-contrôleurs comme unité de traitement pour les capteurs [18, 19], comme la plate-forme Telosb de Crossbow qui est équipée d'un micro-contrôleur MSP 430 à 8 MHz. Ils sont petits, moins chers, et programmables. Grâce à ces caractéristiques techniques, les micro-contrôleurs deviennent un choix idéal pour les systèmes embarqués.

Un défaut important des micro-contrôleurs est le manque de puissance de calcul. Malgré les progrès récents dans le domaine de l'électronique et de l'informatique industrielle, il y a toujours très peu de micro-contrôleurs puissants qui sont développés et commercialisés. Généralement cela est considéré comme un choix du marché, car aujourd'hui dans la plupart de projets, les micro-contrôleurs n'ont pas besoin d'être puissants. Ils sont initialement inventés et mis en place pour effectuer des tâches simples, comme la coupure et la remise de courant dans les circuits intégrés.

Cependant dans les réseaux de capteurs, les fonctionnalités à assurer sont plus com-

pliquées. Dans un vrai réseau, les nœuds doivent mesurer des valeurs environnementales, agréger et compresser les données collectées, même appliquer des mécanismes de sécurité contenant des calculs cryptographiques très complexes. Afin de réaliser une fabrication et un déploiement en grand nombre, il faut que l'unité de traitement soit petite et ait un coût et une consommation d'énergie faibles, mais en même temps, il faut qu'elle puisse gérer beaucoup de tâches différentes dans le réseau.

Pour contourner le problème, maintenant dans les applications existantes, on demande aux nœuds de coopérer entre eux pour atteindre un objectif collectif. Le nœud du même modèle peuvent jouer des rôles différents, ou le même nœud peut assurer plusieurs fonctionnalités pendant des périodes différentes.

2.2.3/ SÉCURITÉ

Comme tous les réseaux informatiques, les réseaux de capteurs font aussi l'objet de différentes attaques. La seule solution est de mettre en place des mécanismes de sécurité qui nous permettent de protéger des informations circulant dans le réseau sans consommer trop de ressources. Les techniques appliquées doivent répondre aux exigences principales suivantes :

– Confidentialité : les informations ne doivent jamais circuler entre les nœuds en claire, notamment dans les applications militaires et de surveillance. Il faut s'assurer que les informations ne sont pas directement interprétables et compréhensibles aux attaquants.
– Intégrité : les données ne doivent pas être altérées pendant la transmission, et il faut que le récepteur puisse vérifier si les données ont subi des altérations.
– Fraîcheur : il s'agit d'un test qui vérifie si les informations reçues sont récentes. Parfois, l'attaquant peut tromper les nœuds en renvoyant des commandes qu'il a intercepté auparavant. Par la fraîcheur de données, le récepteur peut tester si la commande reçue a été réutilisée.
– Authentification : elle nous permet de protégé le réseau contre des attaques de vol d'identité en vérifiant si l'identité déclarée par un nœud est bien celle du nœud déclaré.

Il existe déjà des techniques matures qui peuvent atteindre les objectifs listés ci-dessus, mais la plupart entre elles sont initialement conçues et testées pour les systèmes sans contrainte de ressources. Comme nous avons vu précédemment, les nœuds ne sont pas assez puissants pour gérer des algorithmes et des protocoles très lourds. Il faut les optimiser et modifier pour s'adapter aux réseaux de capteurs.

2.2.4/ TOLÉRANCE AUX PANNES

Les nœuds sont des dispositifs électroniques, qui peuvent tomber en panne pendant leur fonctionnement. Le manque de protection solide et de puissance de calcul rendent les nœuds très fragiles et sensibles face aux dysfonctionnements causés principalement par des attaques physiques, des phénomènes climatiques et géologiques.

Par exemple un glissement de terre peut détruire, enterrer et déplacer des nœuds déployés, et changer complètement la topologie du réseau. Si les nœuds assurant des fonctionnalités particulières sont endommagés, comme les nœuds de passerelle, une

partie du réseau deviendra isolé et inaccessible. L'objectif des techniques de tolérance aux pannes est donc de détecter des anomalies et de restaurer, même partiellement, les fonctionnalités pour que le réseau puisse continuer à fonctionner.

Une solution qui est largement utilisée est la redondance. L'idée est d'avoir plusieurs composants prêts pour assurer la même fonctionnalité. Quant aux réseaux de capteurs, le principe reste le même, on peut attribuer la même tâche à plusieurs nœuds. Lorsqu'un nœud tombe en panne, un nœud de secours va prendre le relais et continuer à travailler sur la tâche. Par exemple, lorsque la pile d'un nœud est épuisée, au lieu de se rendre sur place pour le localiser et le changer, il est plus efficace et moins coûteux de laisser le nœud épuisé et en choisir un autre qui peut reprendre son travail. Les techniques de tolérance aux pannes dans les réseaux de capteurs sont présentées en détail dans le chapitre 5.

2.3/ PROBLÈMES DE SÉCURITÉ DANS LES RÉSEAUX DE CAPTEURS

Les spécificités des réseaux de capteurs sans fil citées précédemment exposent les nœuds à de nombreuses menaces. Il existe principalement 2 types d'attaques, qui sont l'attaque passive et active. Dans le premier cas, si le réseau ne chiffre pas ses données, l'attaquant pourra récupérer les informations qui circulent entre les nœuds en se mettant discrètement dans le réseau et écoutant les communications radio. Dans le cas d'attaque active, l'attaquant est beaucoup plus offensif, au lieu d'espionner discrètement les données, il cherche à perturber le réseau en modifiant les informations ou en endommageant directement les nœuds.

Dans cette section, nous allons voir d'abord une taxonomie des attaques principales au sein des réseaux de capteurs [65], ensuite des mécanismes de sécurité que nous pouvons mettre en place pour les protéger.

2.3.1/ ÉCOUTE PASSIVE

Il s'agit de l'interception et de la collection des données qui circulent dans les réseaux. L'attaquant peut obtenir l'accès à l'ensemble d'informations échangées entre les nœuds, si elles ne sont pas correctement chiffrées. L'objectif de l'écoute passive est l'espionnage des données sensibles sans interférer le fonctionnement du réseau. Par sa nature passif, ce genre d'attaque est très difficile à détecter.

2.3.2/ ANALYSE DU TRAFIC

L'analyse du trafic est une attaque qui est basée sur l'écoute passive, mais au lieu d'espionner directement des données, elle s'intéresse plutôt aux destinations des paquets envoyés. En analysant les chemins empruntés par les paquets, l'attaquant peut étudier la topologie du réseau et identifier les nœuds les plus importants. Ces informations permettent à l'attaquant de préparer des attaques plus précises qui peuvent gravement menacer les communications dans le réseau.

Comme dans le jeu du chasseur de panda [76], les nœuds sont déployés pour tracer la position du panda, et les paquets contenant les informations de localisation sont chiffrées.

Sans accès direct aux données communiquées, l'attaquant peut toujours localiser la zone où se trouve le panda en analysant les chemins de routage des paquets.

Généralement dans un réseau de capteurs, les communications intensives se trouvent souvent soit vers la destinations des paquets, soit près de la source d'événement important. Une analyse du trafic permet à l'attaquant de localiser des point d'intérêt au sein d'un réseau, comme la station de base, les nœuds de passerelles etc.

2.3.3/ Brouillage radio

On utilise souvent le terme anglophone *radio jamming* pour désigner ce genre d'attaque. L'attaquant peut saturer le réseau en diffusant des ondes ayant exactement la même fréquence que celle utilisée par le réseau de capteurs [109]. Les nœuds utilisent l'air comme médium de communication, qui est en effet accessible à tout le monde, l'attaquant peut brouiller le réseau en plaçant simplement quelques nœuds malicieux dans la zone.

2.3.4/ Inondation

L'idée est de saturer ou de perturber le réseau en diffusant massivement des messages inutiles. Les protocoles de routage permettent souvent au réseau d'ajouter de nouveaux nœuds après le déploiement. Quand un nouveau nœud est déployé et réveillé, il envoie périodiquement des message HELLO pour que les autres nœuds qui se situent dans sa portée radio puissent le détecter.

L'attaquant peut déployer des nœuds particuliers ayant une puissance d'émission très forte, qui diffusent constamment des messages HELLO dans le réseau. Lorsqu'un nœud reçoit un HELLO d'un nouvel arrivant, il est obligé de lui renvoyer un message d'accueil qui leur permet de lancer la procédure d'intégration. Si le nœud malicieux répète sans arrêt le message HELLO avec une puissance d'émission assez forte, les autres nœuds dans sa portée radio vont continuer à lui retourner des messages d'accueil (voir figure 2.5).

Par conséquent, ces nœuds deviennent indisponibles, car ils sont occupés pour accueillir le nœud malicieux. En plus, la communication radio consomme beaucoup d'énergie, l'attaquant peut vider les piles des nœuds trompés en leur demandant des messages d'accueil.

2.3.5/ Trou noir

Les nœuds malicieux s'intègrent dans le réseau, et essaient de tromper les autre nœuds en leur faisant croire qu'ils ont découvert un chemin plus fiable. Dans les protocoles de routage, les nœuds évaluent souvent les connexions sans fil avec la puissance des signaux, les nœuds sans protection d'authentification vont abandonner leurs chemins de routage actuels et envoyer les données aux nœuds malicieux qui disposent d'une puissance d'émission plus élevée [42]. De ce fait, les données n'arriveront jamais à la destination prévue (voir figure 2.6).

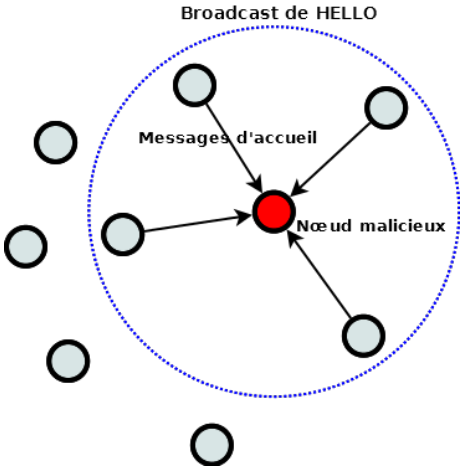


FIGURE 2.5 – Attaque d'inondation de message HELLO

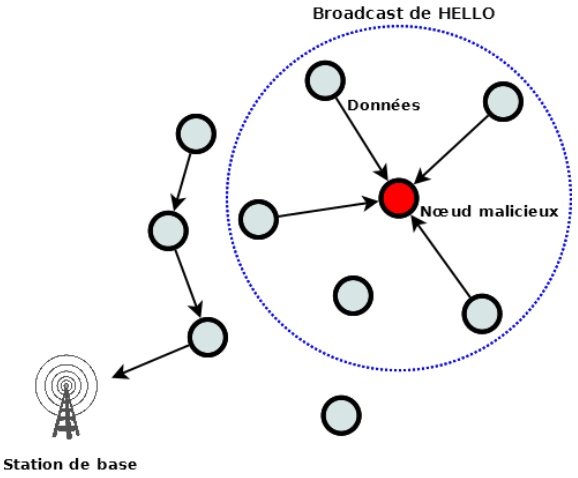


FIGURE 2.6 – Attaque de trou noir

2.3.6/ RÉPLICATION DES DONNÉES

L'attaquant peut réutiliser ultérieurement des paquets qu'il a capturé et enregistré pour tromper le réseau. Même les paquets sont chiffrés et protégés par des mécanismes d'authentification, et l'attaquant n'a aucun accès aux données qu'ils contiennent, on peut toujours deviner l'usage des paquets en observant le comportement des récepteurs.

Par exemple, si l'attaquant s'aperçoit que les nœuds coupent les communications sans fil après la réception d'un paquet, même il ne peut pas lire son contenu, il pourra tout de même déduire que c'est le paquet contenant la commande pour éteindre l'antenne radio. Ensuite l'attaquant peut répliquer et rediffuser le paquet à tous les nœuds autour de lui pour les passer au silence radio [96].

Cette attaque est réalisable si le réseau ne dispose d'aucun mécanisme assurant la fraîcheur des données, ou l'attaquant peut déchiffrer et modifier les contenus des paquets.

2.3.7/ VOL D'IDENTITÉ

Dans les réseaux de capteurs, le vol d'identité s'appelle aussi une attaque sybile [73]. Les nœuds malicieux se déguisent en nœuds légitimes du réseau en utilisant leurs identités. Quand le nœud malicieux est considéré comme un membre du réseau, il est autorisé à participer à toutes les opérations, pendant lesquelles il peut ensuite soit espionner les données, soit éventuellement perturber le réseau.

Un nœud malicieux déguisé peut sévèrement menacer la sécurité du réseau, notamment quand on lui attribue un rôle important, comme le nœud de passerelle qui s'occupe de la communication entre des sous réseaux, ou le cluster-head qui est chargé de la communication inter-cluster. Si le nœud obtient une telle priorité, il pourra prendre des actions qui peuvent gravement interférer le fonctionnement du réseau, comme le refus de routage et la modification de données.

2.3.8/ ATTAQUE PHYSIQUE

L'attaquant se rend physiquement dans la zone où les nœuds sont déployés pour les détruire ou les capturer. Dans les réseaux de capteurs, les nœuds sont déployés aléatoirement pour découvrir uniformément une zone d'intérêt, et il est très difficile de surveiller simultanément chacun des nœuds. La plupart des mécanismes de sécurité sont conçus et appliqués pour protéger les données et leur circulation contre des intrusions et des espionnages, comme le chiffrement de données, l'authentification, le test d'intégrité etc. Cependant les nœuds eux-même restent toujours très fragiles face aux attaques physiques.

Si l'attaquant effectue une analyse du trafic avant de lancer l'attaque, il va pouvoir viser les nœuds les plus importants dans le réseau. Par exemple si l'attaquant réussit à identifier et détruire les nœuds passerelle, le réseau deviendra complètement inaccessible ou sera coupé en plusieurs sous réseaux qui ne peuvent pas communiquer entre eux.

L'attaquant peut aussi capturer et compromettre des nœuds. Un nœud compromis représente une menace très grave pour la sécurité du réseau. L'attaquant peut retrouver des informations sensibles dans la mémoire du nœud capturé, comme les clés cryptographiques, ou modifier carrément le programme installé. Une fois le nœud compromis, l'attaquant le remet dans le réseau pour espionner les données ou interférer les communications [108].

2.3.9/ VARIATIONS DES ATTAQUES

Il existe encore d'autres attaques qui sont en effet des variantes des attaques mentionnées précédemment. On peut faire évoluer leurs fonctionnements ou les combiner ensemble pour en créer une nouvelle. Le but d'une telle variation est de rendre les attaques plus puissantes et précises.

Par exemple l'attaque de trou gris, est basée sur l'attaque de trou noir, qui consiste à tromper les autres nœuds du réseau avec une puissance d'émission élevée. Les données attirées et récupérées par le nœud malicieux sont simplement supprimées et n'arriveront jamais à la destination.

Contrairement au trou noir, un trou gris cherche plutôt à censurer les informations. Les données sont sélectionnées et modifiées avant être renvoyées, ou le nœud malicieux ne laisse passer que des informations moins importantes. Le trou gris peut rendre le réseau beaucoup moins efficace, car les informations importantes sont perdues. Par ailleurs, un trou gris est plus difficile à détecter, car il se comporte comme un nœud normal du réseau.

Parfois l'attaquant peut utiliser en même temps plusieurs techniques d'attaquant différentes. Par exemple, il lance d'abord une analyse du trafic qui lui permet d'identifier les chemins de routage, ainsi que les nœuds de routage qui sont chargés de la transmission de données. Ensuite, il effectue une attaque sybile en déployant des nœuds qui se déguisent en nœud de routage. Au lieu d'envoyer les données vers la station de base, les nœuds malicieux créent une boucle infinie en renvoyant les données vers le début du chemin de routage. Cette attaque peut non seulement perturber le routage des données, mais aussi augmenter considérablement la consommation d'énergie des nœuds qui se situent sur le chemin de routage.

2.4/ MÉCANISMES DE SÉCURITÉ

L'ensemble d'attaques présentées dans la section précédente nous a montré la nécessité de mettre en place des mécanismes de sécurité pour protéger les données et les communications sans fil au sein d'un réseau de capteurs. Néanmoins, à cause de la diversité des attaques et le manque de ressources des nœuds, il est très difficile de proposer une solution qui résolve tous les problèmes. La partie suivante présente des solutions de sécurité répandues en prenant en compte les spécificités des réseaux de capteurs, notamment la consommation d'énergie et la puissance de calcul.

2.4.1/ CLÉ D'AUTHENTIFICATION DYNAMIQUE

Le principe de cette solution est de demander à la station de base de générer et distribuer périodiquement de nouvelles clés d'authentification [6]. A chaque période, la station de base génère une clé qui est ensuite envoyée à l'ensemble du réseau. Les nœuds utilisent cette clé pour prouver leur appartenance au réseau, et un nœud qui ne possède pas la bonne clé n'est pas autorisé à communiquer avec les autres nœuds.

Cette solution offre une protection efficace contre les nœuds malicieux qui essaient de se déguiser en nœuds légitimes, car elle ne demande ni calcul coûteux, ni échange fréquent de données. Même si l'attaquant réussit à capturer et compromettre des nœuds, les nœuds compromis ne pourront toujours pas rentrer dans le réseau, car la nouvelle clé utilisée ne sera plus la même.

Cependant cette solution nécessite une distribution de clés sécurisée et fiable, il ne faut pas qu'un attaquant puisse intercepter et interpréter la nouvelle clé utilisée. D'ailleurs, le réseau est indisponible lors de la distribution de nouvelles clés, car les nœuds ne peuvent pas communiquer entre eux tant que la distribution de clés n'est pas terminée. Un autre inconvénient est qu'elle n'autorise pas l'ajout de nouveaux nœuds, car on ne peut pas connaître à priori la prochaine clé d'authentification à utiliser.

2.4.2/ RÉSEAUX DE CONFIANCE

Cette solution est principalement basée sur l'évaluation de la réputation de chaque nœud. Les nœuds donnent à chacun de leurs voisins une indice de réputation qui peut varier en fonction de son comportement [6, 111, 75, 86, 17, 113]. Pendant les communications sans fil, chaque nœud du réseau surveille les réactions de ses voisins, si un voisin réagit comme il faut, le nœud va incrémenter son indice de réputation. Cependant, s'il détecte des comportements suspects du voisin, comme l'absence ou le retard de réponse, alors le nœud va diminuer l'indice du voisin. Les comportements suspects peuvent être dus à plusieurs causes possibles, par exemple pile épuisée, connexion instable, ou éventuellement nœud malicieux.

Les nœuds du réseau stockent en mémoire un tableau contenant les indices de réputation de tous les voisins, ils choisissent toujours les voisins ayant de meilleures réputations lors du routage de données. Car ils sont considérés comme les voisins les plus sûrs qui peuvent construire un chemin de routage assez fiable.

Cependant, l'utilisation toute seule de cette solution n'est pas suffisante pour protéger le réseau, car elle ne peut pas détecter des attaques passives, comme l'espionnage de données, le nœud réagit correctement comme un nœud légitime, mais toutes les données qui sont passées par lui sont enregistrées localement et accessibles à l'attaquant.

2.4.3/ STÉGANOGRAPHIE

La stéganographie est une solution très simple qui peut être mise en place aisément. Au lieu de rendre le message illisible comme la cryptographie, la stéganographie cherche plutôt à cacher les informations sensibles dans les paquets de contrôle ou les données. Généralement, dans les paquets, notamment la partie d'entête, il existe toujours quelques octets qui ne sont pas utilisés, on peut donc les utiliser pour stocker les données [66]. L'attaquant ne peut pas détecter leur existence tant qu'il ne sait pas la technique exacte utilisée.

La fiabilité de la stéganographie est en effet un pari sur le fait que l'attaquant ne sait pas où se sont cachées les données. Cependant, une fois la technique utilisée dévoilée, l'attaquant pourra accéder à toutes les données cachées. D'ailleurs, le nombre d'octets non utilisés dans des paquets est très limité, on ne peut pas y stocker des informations très volumineuses.

2.4.4/ CRYPTOGRAPHIE

La cryptographie est une solution de sécurité qui est relativement beaucoup plus sûre. L'idée de base est d'utiliser des méthodes mathématiques pour transformer le message original à une suite de données qui ne peuvent pas être directement interprétées par une tierce partie. Elle comprend un ensemble de techniques qui sont fréquemment utilisées dans le monde informatique pour assurer la confidentialité, l'intégrité et l'authenticité des données.

L'application de la cryptographie implique souvent des calculs intensifs et la gestion des données volumineuses, qui ne posent aucun problème pour des plate-formes possédant une puissance de calcul suffisante et un accès de mémoire rapide. Quant à la plupart des

réseaux de capteurs, aucune des conditions n'est satisfaite. La limitation des ressources des nœuds est considérée comme un obstacle principal qui empêche l'utilisation des cryptosystèmes robustes dans les réseaux de capteurs.

La communauté scientifique est toujours à la recherche de solutions qui nous permettent d'optimiser et d'adapter des algorithmes et des protocoles cryptographiques pour les réseaux de capteurs. Le temps de calcul, la consommation de mémoire et d'énergie sont donc devenus des facteurs critiques pour évaluer des solutions proposées.

Il existe principalement 2 catégories de cryptographie, qui sont respectivement la cryptographie symétrique et la cryptographie asymétrique. Cette section est consacrée à la présentation et la comparaison des 2 types de cryptographie, ainsi que leurs avantages et inconvénients.

2.4.4.1/ CRYPTOGRAPHIE SYMÉTRIQUE

La cryptographie symétrique utilise une seule clé pour le chiffrement et le déchiffrement des données, elle est considérée comme une solution moins coûteuse et plus facile à implémenter. Par exemple, la longueur de clé recommandée pour l'algorithme DES est 56 bits, et celle de l'algorithme AES est 128 bits. On suppose que M est le message à chiffrer et k est la clé secrète partagée par l'envoyeur et le récepteur du message. L'envoyeur utilise la fonction de cryptage $E()$ et la clé k pour générer un message chiffré $E(M,k) = M'$. Après la réception du message chiffré, le récepteur utilise une fonction de décryptage $D()$ et la même clé k pour déchiffrer le message $D(M',k) = M$. On a montré dans [52] que l'application des algorithmes de cryptographie symétrique est faisable dans les réseaux de capteurs.

Cependant cette solution possède aussi des inconvénients inhérents. Pour que 2 nœuds puissent communiquer entre eux, il faut qu'ils utilisent exactement la même clé, et la distribution de clés est devenue un défi inévitable. Il existe dans la littérature 4 types de distribution :

- Clé globale : Une seule clé est partagée par tous les nœuds du réseau. C'est la solution la plus facile à mettre en place, et elle offre un minimum de protection contre l'espionnage de données. Cependant, si l'attaque réussit à compromettre un seul nœud, tout le mécanisme de sécurité va échouer.
- Clé partagée par paire de nœuds : Chaque nœud possède une clé unique pour chacun de ses voisins. Une telle gestion de clés peut augmenter considérablement le niveau de sécurité du réseau, car chaque pair de nœuds utilise une clé distincte pour sécuriser la communication entre eux. Si un nœud possède n voisins, il faudra qu'il stocke en mémoire n clés. Premièrement, cette distribution de clés peut engendrer éventuellement des communications radio très fréquentes qui peuvent vider rapidement les piles des nœuds. Deuxièmement, le stockage des clés cause une consommation de mémoire importante.
- Clé individuelle : Chaque nœud détient une clé qui est partagée par lui-même et la station de base, mais cette distribution sécurise seulement la communication entre les nœuds et la station de base. Les nœuds ne peuvent pas communiquer entre eux, car ils ne possèdent pas de clé partagée.
- Clé partagée par groupes de nœuds. Une clé est partagée entre un groupe de nœuds, par exemple, dans un cluster, le cluster-head est chargé de la génération et de la distribution de clé, et la communication intérieure du cluster est sécurisée avec ces

clés partagées. C'est une solution qui a atteint un compromis entre la consommation des ressources et la sécurité.

D'ailleurs, la cryptographie symétrique n'offre pas la signature numérique, sans laquelle les nœuds ne peuvent pas authentifier les messages reçus, et sont vulnérables aux attaques de vol d'identité.

2.4.4.2/ CRYPTOGRAPHIE ASYMÉTRIQUE

La cryptographie asymétrique, appelée aussi cryptographie à clé publique, utilise 2 clés différentes respectivement pour le chiffrement et le déchiffrement. Avant l'envoi du message M, on le chiffre avec la clé publique du destinataire k_p, qui est diffusée librement dans l'ensemble du réseau, $E(M, k_p) = M'$ où $E()$ est la fonction de cryptage et M' est le message chiffré. Après la réception du M', le destinataire utilise sa clé privée k_s, qui est gardée secrète, pour déchiffrer le message, $D(M', k_s) = M$ où $D()$ est la fonction de décryptage.

La cryptographie asymétrique donne aussi la possibilité de signer des messages chiffrés. On suppose que A veut envoyer un message M chiffré et signé à B. A calcule d'abord le condensat du M avec une fonction de hachage $H(M)$, ensuite A génère la signature S_M en chiffrant le condensat avec sa clé privée $S_M = E(k_{s_A}, H(M))$. Enfin, A chiffre la signature S_M et le message original M avec la clé publique de B, $M' = E(k_{p_B}, (S_M, M))$. Quand B reçoit le message chiffré et signé, il déchiffre d'abord M' avec sa clé privée $D(k_{s_B}, M') = (S_M, M)$, ensuite pour vérifier la signature, il déchiffre S_M avec la clé publique de A, $D_{S_M} = D(k_{p_A}, S_M)$ et calcule le condensat de M avec la même fonction de hachage $H(M)$. Enfin, il teste si $D_{S_M} = H(M)$. Si l'authentification échoue, c'est-à-dire $D_{S_M} \neq H(H)$, le message M' sera rejeté.

La signature donne une protection complémentaire au réseau, mais les algorithmes de cryptographie asymétrique sont généralement beaucoup plus coûteux que ceux de cryptographie symétrique, et les clés à manipuler sont plus longues aussi. En pratique, on utilise souvent les 2 solutions en même temps. Par exemple, on utilise la cryptographie asymétrique pour la signature et la génération de clé partagée (voir section 3.4.1 du chapitre 3), ensuite pour le chiffrement et le déchiffrement de données, on utilise la cryptographie symétrique qui offre une rapidité de calcul intéressante.

Concernant l'application de la cryptographie asymétrique dans les réseaux de capteurs, il est évident que les nœuds ne sont pas assez puissants pour gérer directement les algorithmes de cryptographie asymétrique. Les travaux de recherche récents ont montré qu'en faisant des optimisations, il est possible d'implémenter des algorithmes de cryptographie asymétrique pour les réseaux de capteurs [107, 62].

2.5/ CONCLUSION

La cryptographie est la solution la plus utilisée pour sécuriser les communications dans les réseaux. Idéalement, il est préférable d'utiliser la cryptographie asymétrique pour protéger les réseaux de capteurs. Hormis son usage basique, le chiffrement de données, la cryptographie asymétrique offre encore la signature numérique et la génération de clé partagée, qui constituent une protection plus complète.

Néanmoins, par rapport à la cryptographie symétrique, la sécurité basée sur la cryptographie asymétrique implique des calculs beaucoup plus complexes, qui sont dans la plupart des cas très coûteux pour les nœuds. Il existe 2 cryptosystèmes asymétriques principaux, RSA et ECC, dont le premier est le cryptosystème asymétrique le plus utilisé, tandis que le deuxième fait partie des cryptosystèmes asymétriques les plus attractifs.

ECC est un cryptosystème asymétrique basé sur le problème de logarithme discret sur les courbes elliptiques. Il peut offrir le même niveau de sécurité que RSA en utilisant des clés beaucoup plus courtes, et ses spécificités ont encouragé les chercheurs à essayer de l'appliquer dans les réseaux de capteur. Une présentation détaillée de ECC se trouve dans le chapitre 3.

3

CRYPTOGRAPHIE SUR LES COURBES ELLIPTIQUES

Sommaire

La cryptographie sur les courbes elliptiques, elliptic curve cryptography (ECC) en Anglais, est proposée indépendamment par Koblitz [44] et Miller [69] dans les années 80. Elle comprend un ensemble de techniques qui nous permettent de sécuriser des données en consommant moins de ressources. Elle attire récemment de plus en plus d'attention des chercheurs du monde entier, notamment pour le domaine de systèmes embarqués dans lequel les dispositifs électroniques ne possèdent qu'une puissance de calcul très limitée.

L'avantage le plus important d'ECC par rapport aux autres algorithmes de cryptographie asymétrique, par exemple RSA [89], est que l'on peut avoir un bon niveau de sécurité en utilisant une clé beaucoup plus courte. Les tests de performance de Gura et al. [32] ont montré que pour avoir le même niveau de sécurité qu'une clé RSA de 1024 bits, avec ECC, il suffit d'utiliser une clé de 160 bits. D'ailleurs, l'utilisation d'une clé plus courte implique aussi une consommation de mémoire et d'énergie moins importante et un calcul plus rapide.

Pour expliquer le fonctionnement d'ECC, dans ce chapitre nous étudions d'abord les concepts mathématiques fondamentaux de la courbe elliptique, ainsi que l'opération la plus importante sur les courbes, la multiplication scalaire. Ensuite nous présentons l'ensemble d'algorithmes et de techniques proposés dans la littérature qui visent à améliorer la performance de calculs sur les courbes. Enfin nous présentons l'ensemble des protocoles cryptographiques qui sont basés sur les courbes elliptiques.

3.1/ GÉNÉRALITÉS

Avant de présenter les courbes elliptiques, nous étudions d'abord les notions mathématiques dont nous avons besoin pour comprendre son fonctionnement.

3.1.1/ GROUPE

En mathématique, un groupe est un couple $(E, \cdot)$ où E est un ensemble et $\cdot$ est une loi de composition interne qui combine deux éléments a et b de E pour obtenir un troisième élément $a \cdot b$. Il faut que la loi satisfasse les quatre axiomes ci-dessous.

- Fermeture : $\forall (a, b) \in E \mid a \cdot b \in E$
- Associativité : $\forall (a, b) \in E \mid (a \cdot b) \cdot c = a \cdot (b \cdot c)$
- Élément neutre : $\exists e \in E \mid a \cdot e = e \cdot a = a$
- Symétrique : $\forall a \in E, \exists b \in E \mid a \cdot b = b \cdot a = e$

3.1.2/ GROUPE ABÉLIEN

Un groupe abélien, ou un *groupe commutatif*, est un groupe dont la loi de composition interne est commutative. Un ensemble E est un groupe commutatif lorsque

$$\forall (a, b) \in E \mid a \cdot b = b \cdot a$$

3.1.3/ GROUPE CYCLIQUE

Un groupe fini G est cyclique si tout élément du groupe peut s'exprime sous forme d'une puissance ou d'un multiple d'un élément particulier g, appelé le générateur du groupe, c'est-à-dire $G = \langle g \rangle = \{g^n \mid n \in \mathbb{Z}^*\}$. Par exemple si $G = \{g^0, g^1, g^2, g^3, g^4, g^5\}$ et $g^6 = g^0$, alors G est un groupe cyclique. Tout groupe cyclique est abélien car $g^n g^m = g^{n+m} = g^{m+n} = g^m g^n$.

L'ordre d'un élément e d'un groupe cyclique est le nombre entier n positif le plus petit tel que $ne = 0$ (en notation additive) ou $e^n = 1$ (en notation multiplicative). Reprenons le même groupe G du paragraphe précédent, par exemple l'ordre de l'élément g^2 est 3 car l'élément neutre du groupe est $g^0 = 1$ et $(g^2)^3 = g^6 = 1$.

3.1.4/ ANNEAU UNITAIRE ET ANNEAU COMMUTATIF

Un anneau unitaire, ou simplement anneau, est un ensemble E muni de deux lois de composition, notées $+$ (addition) et $\cdot$ (multiplication). E est un anneau, si

- $(E, +)$ est un groupe commutatif
- La loi $\cdot$ est associative et distributive par rapport à la loi $+$.
- La loi $\cdot$ possède un élément neutre.

Il existe un élément neutre de la loi de composition $+$, noté 0 tel que $\forall a \in E \mid a+0 = 0+a = a$ et $\forall a \in E, \exists b \in E \mid a + b = 0$. Il existe un autre élément neutre pour la loi de composition $\cdot$, noté 1 tel que $\forall a \in E \mid a \cdot 1 = 1 \cdot a = a$.

Un anneau commutatif est un anneau dont la loi de composition $\cdot$ est commutative, c'est-à-dire $\forall (a, b) \in E \mid a \cdot b = b \cdot a$.

3.1.5/ CORPS

Un corps est un ensemble E muni de deux lois de composition, notée respectivement $+$ et $\cdot$. Il faut que les deux lois satisfassent les conditions suivantes :

- Le couple $(E, +)$ forme un groupe abélien, il existe un élément neutre, noté 0, tel que $\forall a \in E \mid a + 0 = 0 + a = a$.
- Le couple $(E \backslash \{0\}, \cdot)$ forme aussi un groupe abélien dont l'élément neutre est 1, $\forall a \in E \mid a \cdot 1 = 1 \cdot a = a$.
- La multiplication $\cdot$ est distributif pour l'addition, c'est-à-dire $\forall (a, b, c) \in E \mid a \cdot (b + c) = a \cdot b + a \cdot c$ et $(b + c) \cdot a = b \cdot a + c \cdot a$.

Autrement dit, un corps est un anneau dont les éléments non nuls forment un groupe abélien pour la multiplication.

3.1.6/ CORPS FINI

Un corps fini $\mathbb{F}$ est un corps dont le nombre d'éléments est fini. Le nombre d'éléments est l'ordre du corps, noté q, qui peut être représenté par la puissance d'un nombre premier $q = p^n$, où p est un nombre premier, appelé la caractéristique du corps, et $n \in \mathbb{Z}^+$. Pour

étudier la cryptographie sur les courbes elliptiques, il faut que nous comprenions les deux types de corps ci-dessous.

3.1.6.1/ CORPS PREMIER

Un corps est un corps premier, noté $\mathbb{F}_p$ lorsque l'ordre du corps $q = p^n$ et p est un nombre premier. Le corps est constitué des nombres entiers $\{0, 1, 2, ..., p-1\}$, et $\forall a \in \mathbb{Z}$, $a \bmod p$ donne le reste unique r qui est compris entre $[0, p-1]$.

3.1.6.2/ CORPS BINAIRE

Un corps fini de l'ordre 2^n est un corps binaire, noté $\mathbb{F}_{2^n}$, qui peut être construit en utilisant une représentation polynomiale. Les éléments du corps sont des polynômes binaires dont les coefficients $a_i \in \{0, 1\}$ et les degrés sont inférieurs à n. C'est-à-dire $\mathbb{F}_{2^n} = \{a_{n-1}z^{n-1} + a_{n-2}z^{n-2} + ... + a_1 z + a_0 : a_i \in \{0, 1\}\}$.

3.2/ PRÉSENTATION DE LA COURBE ELLIPTIQUE

Après la présentation des notions mathématiques nécessaires, nous allons passer, dans cette section, à la définition des courbes elliptiques avec l'ensemble d'opérations que nous pouvons effectuer sur elles.

3.2.1/ DÉFINITION DE COURBE ELLIPTIQUE

La courbe elliptique E est une courbe algébrique qui peut être représentée par l'équation Weierstrass (formule 3.1).

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{3.1}$$

On suppose que la courbe est définie dans un corps K et les paramètres $a_1, a_2, a_3, a_4, a_6 \in K$. Pour que la courbe soit lisse et ne contienne aucun point de rebroussement, il faut que le discriminant de la courbe $\Delta \neq 0$. La définition complète de Δ se trouve dans la formule 3.2.

$$\left.\begin{array}{ll} \Delta & = -d_2^2d_8 - 8d_4^3 - 27d_6^2 + 9d_2d_4d_6 \\ d_2 & = a_1^2 + 4a_2 \\ d_4 & = 2a_4 + a_1a_3 \\ d_6 & = a_3^2 + 4a_6 \\ d_8 & = a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2 \end{array}\right\} \tag{3.2}$$

La forme de la courbe peut varier en fonction des paramètres choisis, dans la figure 3.1 nous avons 2 exemples de courbe elliptique.

Dans le domaine cryptographique, nous utilisons les courbes elliptiques qui sont définies dans un corps fini dont l'ordre $q = p^n$. Ce corps peut être soit premier, soit binaire, et le

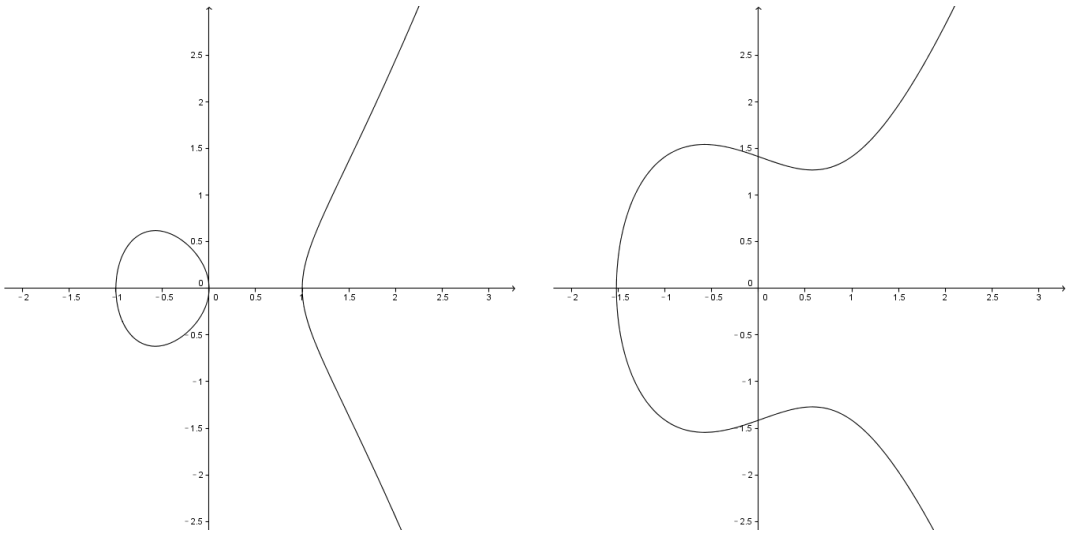


FIGURE 3.1 – Exemples de courbe elliptique

choix de corps n'a pas une influence importante sur la performance du cryptosystème. Dans la littérature, il existe différents algorithmes et techniques qui nous permettent d'optimiser les performances de calcul sur les courbes qui sont définies sur les 2 types de corps [87]. Cependant pour une raison de simplicité d'implémentation et de présentation, durant les travaux de recherche de cette thèse, nous n'avons utilisé que des courbes qui sont définies sur un corps premier.

L'équation Weierstrass d'une courbe elliptique peut être simplifiée, si la courbe est définie sur un corps premier $\mathbb{F}_p$ dont la caractéristique est différente de 2 et de 3. Nous pouvons transformer la formule 3.1 à l'équation de Weierstrass simplifiée (formule 3.3).

$$y^2 = x^3 + ax + b \tag{3.3}$$

où $a, b \in \mathbb{F}_p$. Le discriminant de la courbe $\Delta = -16(4a^3 + 27b^2)$, et $\Delta \neq 0$. C'est aussi la forme de courbe que nous utilisons dans la suite de cette thèse.

3.2.2/ LOI DE COMPOSITION SUR LES COURBES ELLIPTIQUES

Nous supposons qu'une courbe elliptique E est définie dans un corps premier $\mathbb{F}_p$, l'ensemble de points sur la courbe avec le point à l'infini, noté ∞, forment un groupe abélien G dont la loi de composition est l'*addition de point* satisfaisant les conditions suivantes :

- Fermeture : $\forall (P_1, P_2) \in G \mid P_1 + P_2 \in G$.
- Commutativité : $\forall (P_1, P_2) \in G \mid P_1 + P_2 = P_2 + P_1$.
- Associativité : $\forall (P_1, P_2, P_3) \in G \mid (P_1 + P_2) + P_3 = P_1 + (P_2 + P_3)$.
- Élément neutre ∞ : $\forall P \in G \mid P + \infty = \infty + P = \infty$.
- Élément symétrique : $\forall P \in G, \exists Q \in G \mid P + Q = Q + P = \infty$.

Pour obtenir le point symétrique de P, il suffit de changer le signe de sa coordonnée y, si $P = (x, y)$, alors $-P = (x, -y)$ et $P + (-P) = \infty$.

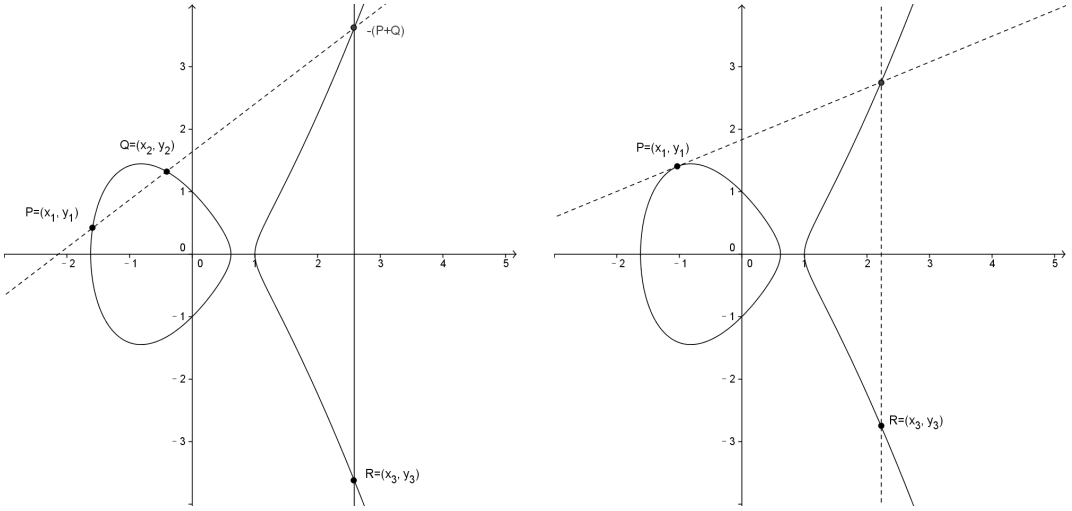
Le calcul de l'addition de point est montré dans les formules 3.4 et 3.5. Supposons que $P = (x_1, y_1) \in E$, $Q = (x_2, y_2) \in E$ et $P \neq \pm Q$, alors $P + Q = (x_3, y_3)$ où

$$x_3 = (\frac{y_2 - y_1}{x_2 - x_1})^2 - x_1 - x_2 \quad \text{et} \quad y_3 = (\frac{y_2 - y_1}{x_2 - x_1})(x_1 - x_3) - y_1 \tag{3.4}$$

Si $P = (x_1, y_1) \in E$ et $P \neq -P$, alors $2P = (x_3, y_3)$ où

$$x_3 = (\frac{3x_1^2 + a}{2y_1})^2 - 2x_1 \quad \text{et} \quad y_3 = (\frac{3x_1^2 + a}{2y_1})(x_1 - x_3) - y_1 \tag{3.5}$$

Une représentation géométrique est donnée dans la figure 3.2. Pour additionner les points P et Q, nous traçons une droite qui passe par ces 2 points, le résultat de l'addition est le point symétrique par rapport à l'axe abscisse du 3e point d'intersection avec la courbe. Pour doubler le point P, c'est-à-dire $2P$, il suffit de trouver la tangente à la courbe au point P, et le résultat du doublement est le point symétrique par rapport à l'axe abscisse du 2e point d'intersection avec la courbe.



Addition : $P + Q = R$ Doublement : $P + P = R$

FIGURE 3.2 – Addition de points sur les courbes elliptiques

3.2.3/ MULTIPLICATION DE POINT

En se basant sur l'addition de point, nous pouvons encore effectuer la *multiplication de point*, notée $Q = kP$ sur une courbe elliptique E où $k \in \mathbb{Z}^+$ et $(P, Q) \in E$. Cette opération est appelée aussi la *multiplication scalaire*, qui est considérée comme l'opération la plus coûteuse et importante sur les courbes elliptiques. Nous allons voir dans la section 3.4 que c'est une opération cruciale dans les protocoles cryptographiques basés sur les courbes elliptiques.

Une multiplication de point peut être considérée comme une suite d'additions de point consécutives :

$$Q = kP = \underbrace{P + P + \cdots + P}_{\text{k fois P}} \tag{3.6}$$

Nous pouvons voir dans les formules 3.4 et 3.5 que le calcul de l'addition de point est compliqué, notamment quand la courbe est définie dans un corps premier, le temps de calcul de l'inverse modulaire (voir section 3.2.4) n'est pas négligeable non plus. Il est donc inefficace de répéter successivement l'addition de point.

La méthode de base pour accélérer la multiplication scalaire est d'utiliser l'algorithme doublement-et-addition (voir algorithme 1). Supposons que P est un point sur une courbe elliptique qui est définie sur un corps premier, notée $E(\mathbb{F}_p)$, pour calculer kP où k est un nombre entier positif de longueur l bits, nous représentons k en binaire $k = \sum_{i=0}^{l-1} k_i 2^i$, et ensuite nous parcourons k du bit de poids faible au bit de poids fort.

Algorithme 1 : Algorithme doublement-et-addition pour calculer $Q = kP$

Données : $k = \sum_{i=0}^{l-1} k_i 2^i$ et $P \in E(\mathbb{F}_p)$
Résultat : $Q = kP$

```
Q ← ∞;
pour i de 0 à l − 1 faire
    si k_i = 1 alors
        Q ← Q + P
    fin
    P ← 2P
fin
retourner Q
```

3.2.4/ ALGORITHME D'EUCLIDE ÉTENDU

Nous travaillons avec des courbes qui sont définies dans un corps premier $\mathbb{F}_p$, les calculs modulaires sont donc indispensables pour limiter les coordonnées des points dans l'intervalle $[0, p-1]$. Les formules de l'addition de point peuvent être transformées sous forme de congruence. Nous avons maintenant

$$x_3 \equiv (\frac{y_2 - y_1}{x_2 - x_1})^2 - x_1 - x_2 \pmod{p} \qquad y_3 \equiv (\frac{y_2 - y_1}{x_2 - x_1})(x_1 - x_3) - y_1 \pmod{p} \tag{3.7}$$

pour l'addition et

$$x_3 \equiv (\frac{3x_1^2 + a}{2y_1})^2 - 2x_1 \pmod{p} \qquad y_3 \equiv (\frac{3x_1^2 + a}{2y_1})(x_1 - x_3) - y_1 \pmod{p} \tag{3.8}$$

pour le doublement.

Nous constatons que dans les formules 3.7 et 3.8 nous avons besoin d'effectuer des calculs de l'inverse modulaire $m \equiv x^{-1} \pmod{p}$ dans un corps premier, noté $\mathbb{Z}/p\mathbb{Z}$. L'inverse modulaire existe si et seulement si le Plus Grand Commun Diviseur $PGCD(x, p) = 1$. Nous pouvons aussi représenter cette condition en utilisant le théorème de Bézout

$$ux + vp = PGCD(x, p) = 1 \tag{3.9}$$

où u est l'inverse multiplicatif de x modulo p, et u, v sont les coefficients de Bézout.

Afin de calculer l'inverse modulaire, nous pouvons utiliser l'algorithme d'Euclide étendu (voir l'algorithme 2) qui est en effet une variante de l'algorithme d'Euclide, et qui permet non seulement de calculer le PGCD de x et p, mais aussi de déterminer les coefficients de Bézout u et v. Un exemple est donné dans l'annexe .4 pour vous montrer le déroulement de l'algorithme.

Le calcul de l'inverse modulaire est l'opération la plus coûteuse durant des multiplications scalaires. Dans un véritable cryptosystème d'ECC, la longueur des nombres entiers manipulés peut atteindre facilement quelques centaines de bits, et nous pouvons constater dans l'algorithme 1 que durant une multiplication scalaire, nous avons besoin de calculer au moins 2 inverses modulaires par itération. Nous allons voir dans la section 3.3 des solutions qui nous permettent d'éviter de répéter le calcul de l'inverse modulaire pendant les multiplications scalaires.

Algorithme 2 : Algorithme d'Euclide étendu

Données : $(x, p) \in \mathbb{Z}$
Résultat : $x^{-1} \pmod{p}$

$(r_1, r_2, u_1, u_2, q, rs, us) \in \mathbb{Z}$;
$r_1 \leftarrow x$;
$r_2 \leftarrow p$;
$u_1 \leftarrow 1$;
$u_2 \leftarrow 0$;
tant que $r_2 \neq 0$ **faire**
    $q \leftarrow \frac{r_1}{r_2}$;
    $rs \leftarrow r_1$;
    $us \leftarrow u_1$;
    $r_1 \leftarrow r_2$;
    $u_1 \leftarrow u_2$;
    $r_2 \leftarrow rs - q \times r_1$;
    $u_2 \leftarrow us - q \times u_1$;
fin
si $r_1 \neq 1$ **alors**
    retourner *Erreur*;
sinon
    retourner u_1;
fin

3.2.5/ PROBLÈME DU LOGARITHME DISCRET SUR LES COURBES ELLIPTIQUES

Nous avons vu les différentes opérations que nous pouvons effectuer sur les courbes elliptiques, notamment la multiplication scalaire qui est utilisée fréquemment pendant les calculs cryptographiques. Dans cette section, nous expliquons la raison pour laquelle la multiplication scalaire est la clé pour assurer la sécurité d'un cryptosystème d'ECC.

Le niveau de sécurité d'ECC dépend fortement de la difficulté pour résoudre le problème de logarithme discret sur les courbes elliptiques. Nous supposons que g est le générateur d'un groupe cyclique G de l'ordre n, si la loi de composition du groupe est multiplicative, tout élément e peut être écrit sous forme $e = g^k$ où $k \in \mathbb{Z}$. En outre deux entiers

quelconques e et k satisfaisant cette équation sont nécessairement congrus modulo n, c'est-à-dire $e \equiv k \pmod{n}$. Le problème du logarithme discret est en effet une application réciproque de g^k (voir la formule 3.10), qui a pour objectif de trouver le plus petit nombre entier naturel k vérifiant cette propriété.

$$\log_g : G \to \mathbb{Z}_n \tag{3.10}$$

Nous choisissons, sur une courbe elliptique définie dans un corps premier fini $E(\mathbb{F}_p)$, un point P de l'ordre n comme le générateur du groupe cyclique $\langle P \rangle$, un autre point $Q \in \langle P \rangle$. Le problème du logarithme discret sur les courbes elliptiques (ECDLP en Anglais) est de trouver $l \in [0, n-1]$ satisfaisant $Q = kP$.

La solution la plus naïve pour résoudre ce problème est de calculer exhaustivement $1P, 2P, 3P \ldots$ jusqu'à ce que nous trouvions Q, mais le calcul peut devenir extrêmement long si la valeur de k est suffisamment grande (voir tableau 3.1). Il est donc énormément difficile de retrouver la valeur de k à partir de Q et P. Il n'y a pas de preuve mathématique qui peut démontrer que le ECDLP est insoluble, mais la résolution d'un tel problème est toujours considéré comme infaisable en prenant compte de l'état actuel des technologies informatiques [34].

Paramètre de courbe	Valeur recommandée
$a =$	-3
$b =$	`0x 64210519 E59C80E7 0FA7E9AB 72243049 FEB8DEEC C146B9B1`
$p =$	$2^{192} - 2^{64} - 1$
$x_G =$	`0x 188DA80E B03090F6 7CBF20EB 43A18800 F4FF0AFD 82FF1012`
$y_G =$	`0x 07192B95 FFC8DA78 631011ED 6B24CDD5 73F977A1 1E794811`
$n =$	`0x FFFFFFFF FFFFFFFF FFFFFFFF 99DEF836 146BC9B1 B4D22831`

TABLE 3.1 – Courbe $E(\mathbb{F}_p) : y^2 = x^3 + ax + b$ avec un générateur $G(x_G, y_G)$ de l'ordre n

3.3/ OPTIMISATION DE LA PERFORMANCE DE MULTIPLICATION SCALAIRE

Un des avantages d'ECC est la rapidité de calcul car nous utilisons des clés plus courtes. ECC est souvent un choix adapté pour les matériels qui ne disposent que d'une mémoire et d'une puissance de calcul limitées (carte à puce, micro-contrôleur etc.). Comme nous avons vu dans les sections précédentes que l'opération la plus complexe sur les courbes elliptiques est la multiplication scalaire, elle est aussi fréquemment utilisée dans les protocoles cryptographiques (voir section 3.4). La performance de la multiplication scalaire a une forte influence sur la performance de l'ensemble de cryptosystème.

Dans cette section nous allons voir de différentes techniques mathématiques et algorithmiques qui nous permettent d'accélérer le calcul de multiplication scalaire.

3.3.1/ MÉTHODE NAF

Comme nous avons déjà précisé, dans cette thèse, nous travaillons avec des courbes elliptiques $E(\mathbb{F}_p)$ qui sont définies dans un corps premier fini dont le caractère est supérieur

à 3. Si $P = (x, y)$ est un point sur $E(\mathbb{F}_p)$, alors $-P = (x, -y)$. Ainsi, la soustraction de point est autant efficace que l'addition.

Nous pouvons voir dans l'algorithme 1 que le temps de calcul dépend principalement du nombre de bits non-nuls dans k. Si nous pouvons réduire le nombre de bits non-nuls, nous pourrons accélérer le calcul.

La première méthode est d'utiliser la forme non adjacente (NAF) qui consiste à représenter un nombre entier positif k de longueur l avec des chiffres binaires signés. C'est-à-dire $k = \sum_{i=0}^{l} k_i 2^i$ où $k_i \in \{0, \pm 1\}$ et $k_{l-1} \neq 0$. En outre, nous ne pouvons pas trouver deux k_i non-nuls consécutifs.

La représentation NAF d'un nombre k dispose des propriétés suivantes :

- k a une représentation unique de NAF, notée $NAF(k)$.
- $NAF(k)$ a moins de bits non-nuls par rapport à toute autre représentation utilisant des chiffres binaires signés.
- La longueur maximale de $NAF(k)$ est $l + 1$ bits.
- Si la longueur de $NAF(k) = l$, alors $\frac{2^l}{3} < k < \frac{2^{l+1}}{3}$.
- Par rapport à la représentation binaire de k, le nombre de bits non-nuls est réduit de $\frac{l}{2}$ à $\frac{l}{3}$.

$NAF(k)$ peut être calculé efficacement avec l'algorithme 3.

Algorithme 3 : Calcul de la forme NAF d'un nombre entier positif

Données : $k = \sum_{i=0}^{l-1} k_i 2^i$ et $k \in \mathbb{Z}^+$
Résultat : $NAF(k)$

```
i ← 0;
tant que k ≥ 1 faire
    si k est impair alors
        k_i ← 2 − (k mod 4);
        k ← k − k_i;
    sinon
        k_i ← 0;
    fin
    k ← k/2;
    i ← i + 1;
fin
retourner (k_{i−1}, k_{i−2}, ..., k_1, k_0)
```

L'algorithme 1 doit aussi être modifié (voir algorithme 4) pour pouvoir prendre en charge des nombres entiers représentés en forme NAF.

3.3.2/ MÉTHODE DE LA FENÊTRE

Si nous disposons de mémoire suffisante, nous pouvons encore optimiser la performance de l'algorithme 4 en utilisant la méthode de la fenêtre.

Une représentation NAF de taille w d'un nombre entier positif k, notée $NAF_w(k)$, est une expression $k = \sum_{i=0}^{l-1} k_i 2^i$ où chaque chiffre k_i non-nul est impair, $|k_i| < 2^{w-1}$ et $k_{l-1} \neq 0$. Il y a au plus un chiffre sur w chiffres est non-nul.

Algorithme 4 : Méthode NAF pour le calcul de multiplication scalaire

Données : $NAF(k)$ où $k \in \mathbb{Z}^+$, et $P \in E(\mathbb{F}_p)$
Résultat : $k.P$

```
Q ← ∞;
pour i de l − 1 à 0 faire
    Q ← 2.Q;
    si k_i = 1 alors
        Q ← Q + P;
    fin
    si k_i = −1 alors
        Q ← Q − P;
    fin
fin
retourner Q
```

La forme $NAF_w(k)$ doit satisfaire des conditions suivantes :

- Un nombre entier k a une représentation unique de $NAF_w(k)$.
- $NAF_2(k) = NAF(k)$.
- La longueur maximale de $NAF_w(k)$ est $l + 1$.
- Le nombre moyen de bits non-nuls est $\frac{l}{w+1}$.

Le calcul de la forme $NAF_w(k)$ est montré dans l'algorithme 5.

Algorithme 5 : Calcul de la forme $NAF_w(k)$ d'un nombre entier positif

Données : k, w
Résultat : $NAF_w(k)$

```
i ← 0;
tant que k ≥ 1 faire
    si k est impair alors
        k_i ← k mod 2^w;
        k ← k − k_i;
    sinon
        k_i ← 0;
    fin
    k ← k/2;
    i ← i + 1;
fin
retourner (k_{i−1}, k_{i−2}, ..., k_1, k_0)
```

Le calcul de multiplication scalaire en utilisant la méthode de la fenêtre est donné dans l'algorithme 6.

Nous pouvons constater qu'avant de lancer la boucle pour parcourir les k_i, il faut d'abord calculer et stocker en mémoire $P_i = i.P$ où $i \in \{1, 3, 5, \ldots 2^{w-1} - 1\}$. Donc cette méthode est applicable si et seulement si le système possède assez de mémoire pour stocker temporairement l'ensemble de P_i précalculés.

Algorithme 6 : Méthode $NAF_w(k)$ pour le calcul de multiplication scalaire

Données : $NAF_w(k), P \in E(\mathbb{F}_p)$
Résultat : $k.P$

Calculer $P_i = i.P$ où $i \in \{1, 3, 5, \ldots 2^{w-1} - 1\}$;
$Q \leftarrow \infty$;
pour i *de* $l - 1$ *à* 0 **faire**
    $Q \leftarrow 2Q$;
    si $k_i \neq 0$ **alors**
        si $k_i > 0$ **alors**
            $Q \leftarrow Q + P_{k_i}$;
        sinon
            $Q \leftarrow Q - P_{-k_i}$;
        fin
    fin
fin
retourner Q

3.3.3/ SYSTÈME DE COORDONNÉES PROJECTIVES

Le calcul dont nous avons besoin pour additionner 2 points en coordonnées affines est détaillé dans les formules 3.4 et 3.5. Nous pouvons remarquer que dans les 2 formules, nous avons besoin d'effectuer des calculs de l'inverse modulaire, qui demande beaucoup plus de calculs supplémentaires (voir algorithme 2). Afin d'éviter la répétition du calcul de l'inverse modulaire, il est plus avantageux d'utiliser des coordonnées projectives.

Avec le système de coordonnées projectives, un point est représenté par 3 coordonnées (X, Y, Z) qui correspondent à un point affine $(\frac{X}{Z^c}, \frac{Y}{Z^d})$, et il est équivalent à un point projectif $(\frac{X}{Z^c}, \frac{Y}{Z^d}, 1)$.

Dans un système de coordonnées projectives standards, $c = 1$ et $d = 1$, mais nous travaillons souvent avec un cas particulier, le système de coordonnées jacobiennes dans lequel $c = 2$ et $d = 3$, et l'équation Weierstrass simplifiée (voir formule 3.3) d'une courbe elliptique peut être transformée en

$$Y^2 = X^3 + aXZ^4 + bZ^6 \tag{3.11}$$

Le point à l'infini est $(1, 1, 0)$, et la négation d'un point (X, Y, Z) est $(X, -Y, Z)$.

3.3.3.1/ DOUBLEMENT DE POINT

Nous supposons que $P = (X_1, Y_1, Z_1) \in E$ et $P \neq -P$. Il nous suffit de réutiliser la formule 3.5 pour calculer $2P = (X_3', Y_3', 1)$, car le point $P = (X_1, Y_1, Z_1)$ est équivalent à $(\frac{X_1}{Z_1^2}, \frac{Y_1}{Z_1^3})$ en coordonnée affine. Nous substituons donc, dans la formule 3.5, la variable x par $\frac{X_1}{Z_1^2}$ et y par $\frac{Y_1}{Z_1^3}$, et nous obtenons :

$$X_3' = (\frac{3\frac{X_1^2}{Z_1^4} + a}{2\frac{Y_1}{Z_1^3}}) - 2\frac{X_1}{Z_1^2} = \frac{(3X_1^2 + aZ_1^4)^2 - 8X_1Y_1^2}{4Y_1^2Z_1^2} \tag{3.12}$$

et

$$Y_3' = (\frac{3\frac{X_1^2}{Z_1^4} + a}{2\frac{Y_1}{Z_1^3}})(\frac{X_1}{Z_1^2} - X_3') - \frac{Y_1}{Z_1^3} = \frac{3X_1^2 + aZ_1^4}{2Y_1Z_1}(\frac{X_1}{Z_1^2} - X_3') - \frac{Y_1}{Z_1^3} \tag{3.13}$$

Enfin pour éliminer les dénominateurs, nous calculons le point (X_3, Y_3, Z_3) dont $X_3 = X_3'.Z_3^2$, $Y_3 = Y_3'.Z_3^3$ et $Z_3 = 2Y_1Z_1$. Le point (X_3, Y_3, Z_3) est équivalent à $(X_3', Y_3', 1)$, car dans un système de coordonnées projectives, (X_1, Y_1, Z_1) est équivalent à (X_2, Y_2, Z_2), si $X_1 = \lambda^c X_2$, $X_1 = \lambda^d Y_2$ et $Z_1 = \lambda Z_2$. Les nouvelles coordonnées sont données dans la formule 3.14 :

$$\left.\begin{array}{ll} X_3 & = (3X_1^2 + aZ_1^4)^2 - 8X_1Y_1^2 \\ Y_3 & = (3X_1^2 + aZ_1^4)(4X_1Y_1^2 - X_3) - 8Y_1^4 \\ Z_3 & = 2Y_1Z_1 \end{array}\right\} \tag{3.14}$$

Nous pouvons remarquer que si le paramètre de la courbe $a = -3$, $(3X_1^2 - 3Z_1^4)$ peut être factorisée sous la forme suivante :

$$3X_1^2 - 3Z_1^4 = 3(X_1 - Z_1^2)(X_1 + Z_1^2) \tag{3.15}$$

qui peut être calculée beaucoup plus efficacement. C'est aussi la raison pour laquelle dans les configurations recommandées de courbes elliptiques, on choisit toujours la valeur -3 pour le paramètre a.

3.3.3.2/ ADDITION DE POINT

Nous avons deux points $P = (X_1, Y_1, Z_1) \in E$ où $Z_1 \neq 0$, et $Q = (X_2, Y_2, 1)$. Afin de simplifier l'implémentation de l'algorithme, la coordonnée Z du point Q est toujours égale à 1. Nous supposons que $P \neq \pm Q$, et nous appliquons la même technique pour le doublement. Comme le point P est équivalent à $(\frac{X_1}{Z_1^2}, \frac{Y_1}{Z_1^3})$ en coordonnée affine, il nous suffit de substituer, dans la formule 3.4, la variable x_1 par $\frac{X_1}{Z_1^2}$ et y_1 par $\frac{Y_1}{Z_1^3}$, et nous obtenons donc les nouvelles formules de calcul (voir les formules 3.16 et 3.17).

$$X_3' = (\frac{Y_2 - \frac{Y_1}{Z_1^3}}{X_2 - \frac{X_1}{Z_1^2}})^2 - \frac{X_1}{Z_1^2} - X_2 = (\frac{Y_2Z_1^3 - Y_1}{(X_2Z_1^2 - X_1)Z_1})^2 - \frac{X_1}{Z_1^2} - X_2 \tag{3.16}$$

et

$$Y_3' = (\frac{Y_2 - \frac{Y_1}{Z_1^3}}{X_2 - \frac{X_1}{Z_1^2}})(\frac{X_1}{Z_1^2} - X_3') - \frac{Y_1}{Z_1^3} = (\frac{Y_2Z_1^3 - Y_1}{(X_2Z_1^2 - X_1)Z_1})(\frac{X_1}{Z_1^2} - X_3') - \frac{Y_1}{Z_1^3} \tag{3.17}$$

Comme dans le calcul de l'addition, pour éliminer les dénominateurs, nous devons calculer le point équivalent du $(X_3', Y_3', 1)$. C'est-à-dire (X_3, Y_3, Z_3) où $X_3 = X_3'.Z_3^2$, $Y_3 = Y_3'.Z_3^3$ et $Z_3 = (X_2Z_1^2 - X_1)Z_1$, nous obtenons ensuite les formules des nouvelles coordonnées :

$$\left.\begin{array}{ll} X_3 & = (Y_2Z_1^3 - Y_1)^2 - (X_2Z_1^2 - X_1)^2(X_1 + X_2Z_1^2) \\ Y_3 & = (Y_2Z_1^3 - Y_1)(X_1(X_2Z_1^2 - X_1)^2 - X_3) - Y_1(X_2Z_1^2 - X_1)^3 \\ Z_3 & = (X_2Z_1^2 - X_1)Z_1 \end{array}\right\} \quad (3.18)$$

Dans le tableau 3.2, nous avons une liste de nombres d'opérations nécessaires pour calculer une addition et un doublement sur les courbes elliptiques. Nous pouvons voir que le système de coordonnées jacobiennes offre une meilleure performance pour le doublement de point, et une addition utilisant des systèmes de coordonnées mixtes (Jacobienne + Affine) demande moins d'opérations.

Nous considérons que le calcul de l'inversion modulaire est une opération beaucoup plus coûteuse que les autres. Cependant, si nous utilisons les systèmes de coordonnées projectives, nous pourrons éviter de répéter le calcul de l'inverse modulaire durant la multiplication scalaire. Il nous suffit de l'appliquer une seule fois à la fin de de la multiplication pour convertir le résultat final en coordonnées affines.

Doublement		Addition		Addition mixte	
$2A \to A$	$1I, 2M, 2S$	$A + A \to A$	$1I, 2M, 1S$	$J + A \to J$	$8M, 3S$
$2P \to P$	$7M, 3S$	$P + P \to P$	$12M, 2S$		
$2J \to J$	$4M, 4S$	$J + J \to J$	$12M, 4S$		

TABLE 3.2 – Nombre d'opérations pour calculer une addition et un doublement sur $y^2 = x^3 - 3x + b$. A : Affine, P : Projective standard, J : Jacobienne, I : Inversion modulaire, M : Multiplication, S : Carré

3.3.4/ RÉDUCTION DE SCALAIRE

Selon l'algorithme 1, lorsque l'on effectue la multiplication scalaire kP, on doit parcourir les bits du scalaire k. Le nombre d'itération dépend du nombre de digits dans la représentation binaire du scalaire. Autrement dit, si on peut trouver un scalaire équivalent plus petit, il sera possible d'accélérer le calcul de multiplication scalaire en substituant la valeur du scalaire par la nouvelle.

Dans [26], Faye et al. ont proposé une méthode qui permet de remplacer la multiplication kP par une représentation équivalente dP, où P est le point de générateur dont $ord(P) = n$, $k > d$ et $k, d \in [\lfloor n/2 \rfloor + 1, n - 1]$. On peut obtenir la valeur de d en suivant les règles ci-dessous :

$$\left\{\begin{array}{ll} \forall k > n, & \exists d = (k - \lfloor k/2 \rfloor n), kP = dP \\ \forall k \in]\lfloor n/2 \rfloor, n - 1], & \exists d = k - n, kP = dP \\ \forall k \in]0, \lfloor n/2 \rfloor], & \exists d = k, kP = dP \\ \forall k \in \{-n, 0, n\}, & kP = \infty \\ \forall k \in [-(n-1), -\lfloor n/2 \rfloor[, & \exists d = (n + k), kP = dP \\ \forall k \in [-\lfloor n/2 \rfloor, 0[, & \exists d = k, kP = dP \\ \forall k < -n, & \exists d = k + n\lfloor |k|/2 \rfloor, kP = dP \end{array}\right. \quad (3.19)$$

En cryptographie, la valeur de k est souvent choisie dans l'intevalle $]0, n-1]$, donc la formule 3.19 peut être simplifiée de la manière suivante :

$$\begin{cases} \forall k \in]\lfloor n/2 \rfloor, n-1], & \exists d = k-n, kP = dP \\ \forall k \in]0, \lfloor n/2 \rfloor], & \exists d = k, kP = dP \end{cases} \tag{3.20}$$

Prenons un exemple, $E : y^2 = x^3 + x + 1$ est une courbe elliptique définie dans un corps premier fini dont le caractéristique $p = 23$, notée $E(\mathbb{F}_{23})$. $P(0, 1)$ est le point générateur, et les représentations équivalentes des multiplications scalaires sont illustrées dans la figure 3.3.

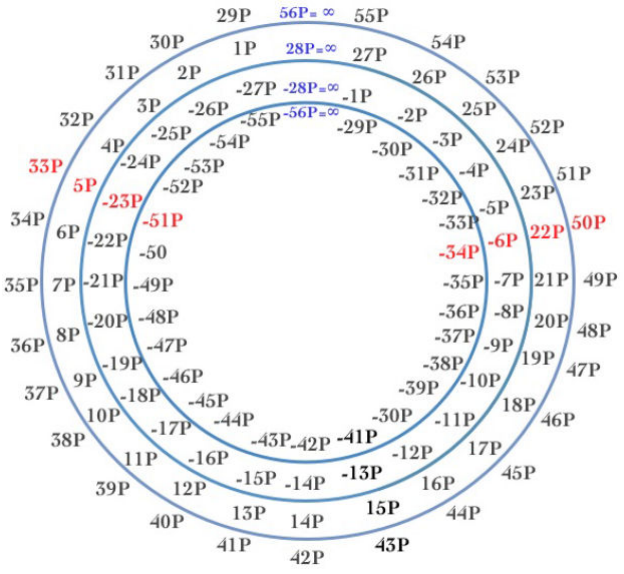


FIGURE 3.3 – Exemples des multiplications scalaires équivalentes

Nous pouvons constater que $-34P$, $-6P$, $22P$ et $50P$ sont équivalentes, si nous voulons calculer le point $50P$, il suffira de calculer $6P$ et changer le signe de sa coordonnée en ordonnée. Les tests de performance ont montré que cette technique peut réduire le temps de calcul, notamment quand la valeur de k est proche de la borne $n-1$.

3.3.5/ MULTIPLICATION SCALAIRE PARALLÈLE

Une autre technique qui nous permet d'accélérer le calcul des multiplications scalaires est le calcul parallèle. Le principe est découper le calcul en plusieurs sous tâches indépendantes qui peut être effectuée séparément par des processeurs différents. Le calcul parallèle des multiplications scalaires est un sujet de recherche très actif dans le domaine de cryptographie, différentes solutions ont été proposées, mais beaucoup entre elles sont conçues pour des implémentations matérielles basées sur les architectures multi-processeurs ou FPGA [80, 77].

Il existe principalement 2 types d'implémentation logicielle, le premier est basé sur la décomposition de scalaire. Nous voulons calculer $Q = kP$, et nous décomposons d'abord

le scalaire k en plusieurs parties, chacune représente une sous tâche de calcul. Elles sont ensuite distribués aux processeurs différents, qui les traitent simultanément.

Dans [8], Brickell et al. ont proposé une méthode d'exponentiation rapide basée sur le tableau précalculé. Par exemple, nous voulons calculer g^n où g est un élément de Z/pZ et n est un nombre entier positif de longueur l. Nous pouvons représenter l'exposant n en base h de la manière suivante :

$$n = \sum_{i=0}^{l-1} a_i x_i \text{ où } 0 \leq a_i \leq h-1 \text{ et } 0 \leq i < l \tag{3.21}$$

Nous calculons et stockons $g^{x_0}, g^{x_1}, \ldots, g^{x_{i-1}}$ dans un tableau, ainsi g^n peut être calculé en utilisant

$$g^n = \prod_{d=1}^{h-1} c_d^d \qquad c_d = \prod_{i:a_i=d} g^{x_i} \tag{3.22}$$

Le calcul de g^n est constitué de trois étapes principales :

1. Déterminer la représentation de $n = \sum_{i=0}^{l-1} a_i x_i$ en base h.
2. Calculer $c_d = \prod_{i:a_i=d} g^{x_i}$.
3. Calculer $g^n = \prod_{d=1}^{h-1} c_d^d$.

Cette méthode peut être appliquée aux multiplications scalaires sur les courbes elliptiques en calculant des points $2^1P, 2^2P, \ldots, 2^{l-1}P$.

Les étapes 2 et 3 peuvent être parallélisée. Par exemple, si nous possédons $h-1$ processeurs, alors nous pouvons paralléliser la 2e étape, et chaque processeur calcule sa propre valeur de c_d. Idem pour l'étape 3, si nous avons d processeurs disponibles, chacun peut calculer c_d^d pour une valeur de d particulière.

Lim and Lee ont proposé dans [60] une autre méthode d'exponentiation rapide. Afin de calculer g^R, où R est un nombre entier positif de longueur n, on divise R en h parties R_i de longueur $\lfloor n/h \rfloor$. Ensuite chaque R_i est encore divisé en v segments $R_{i,j}$ de longueur $\lfloor b = a/v \rfloor$ (voir la formule 3.23).

$$\begin{array}{rll} R & = R_{h-1}R_{h-2}\ldots R_1R_0 & = \sum_{i=0}^{h-1} R_i 2^{ia} \\ \text{où } R_i & = R_{i,v-1}R_{i,v-2}\ldots R_{i,1}R_{i,0} & = \sum_{j=0}^{v-1} R_{i,j} 2^{jb} \end{array} \tag{3.23}$$

Nous supposons que $g_0 = g$, ainsi $g_i = g_{i-1}^{2^a} = g^{2^{ia}}$ où $0 < i < h$. En utilisant la formule 3.23, g^R peut être réécrite de la manière suivante

$$g^R = \prod_{i=0}^{h-1} g_i^{R_i} = \prod_{j=0}^{v-1} \prod_{i=0}^{h-1} (g_i^{2^{jb}})^{R_{i,j}} \tag{3.24}$$

Si la représentation binaire de R_i est $R_i = e_{i,a-1}e_{i,a-2}\ldots e_{i,0}$ et celle de $R_{i,j}$ est $R_{i,j} = e_{i,jb+b-1}e_{i,jb+b-2}\ldots e_{i,jb+1}e_{i,jb}$, alors la formule 3.24 peut être transformée à la forme suivante

$$g^R = \prod_{k=0}^{b-1}(\prod_{j=0}^{v-1}\prod_{i=0}^{h-1} g_i^{2^{jb}e_{i,jb+k}})^{2^k} \tag{3.25}$$

Si nous calculons et stockons les valeurs suivantes pour tout $i \in [1, 2^h[$ et $j \in [0, v[$,

$$\begin{aligned} G[0][i] &= g_{h-1}^{e_{h-1}} g_{h-2}^{e_{h-2}} \cdots g_0^{e_0} \\ G[j][i] &= (G[j-1][i])^{2^b} = (G[0][i])^{2^{jb}} \end{aligned} \tag{3.26}$$

la formule 3.25 peut encore être réécrite comme ci-dessous :

$$g^R = \prod_{k=0}^{b-1}(\prod_{j=0}^{v-1} G[j][I_{j,k}])^{2^k} \tag{3.27}$$

où $I_{j,k} = e_{h-1,bj+k}e_{h-2,bj+k}\ldots e_{1,bj+k}e_{0,bj+k}$ et $j \in [0, b[$. Ainsi, le calcul de g^R peut être parallélisé un utilisant les valeurs précalculées.

Une autre méthode basée sur des valeurs précalculés est proposée dans [71]. Nous supposons que nous voulons calculer $Q = kP$ où Q et P sont 2 points représentés en coordonnées jacobiennes sur une courbe elliptique, et k est un nombre entier positif de longueur 160.

Nous préparons un tableau contenant 62 points précalculés (voir la formule 3.28).

$$\begin{aligned} A[s] &= \textstyle\sum_{j=0}^{4} a_{s,j} 2^{32j} G \\ B[s] &= \textstyle\sum_{j=0}^{4} a_{s,j} 2^{16+32j} G \end{aligned} \tag{3.28}$$

où $1 \le s \le 31$ et $a_{s,0}, \ldots, a_{s,4}$ est la représentation binaire de $s = \sum_{j=0}^{4} a_{s,j} 2^j$. Le calcul parallèle de kP en utilisant des points précalculés est montré dans l'algorithme 7.

Algorithme 7 : Multiplication scalaire parallèle basée sur des points précalculés

Données : $k = \sum_{i=0}^{l-1} k_i 2^i$, P
Résultat : kP

pour $0 \le j \le 15$ **faire**
  $u_j = \sum_{i=0}^{4} k_{32i+j} 2^i$;
  $v_j = \sum_{i=0}^{4} k_{32i+16+j} 2^i$;
fin
$A[0] = \infty$;
$B[0] = \infty$;
$T = \infty$;
pour i *de* 15 *à* 0 **faire**
  $T \leftarrow 2T$;
  $T \leftarrow T + A[u_i] + B[v_i]$;
fin
return T;

Comme cette méthode est basée sur des points précalculés aussi, une fois les points prêts, le calcul qui se situe à l'intérieur de la boucle peut être effectué par des processeurs différents.

Le deuxième type d'implémentation logicielle de multiplication scalaire parallèle est basé sur la décomposition d'opération. C'est-à-dire que chaque processeur s'occupe d'un type d'opération spécifique. Cependant, ce genre d'implémentation nécessite souvent une mémoire partagée entre les processeurs.

Dans [4], une architecture de double-processeur est proposée (voir la figure 3.4) pour paralléliser le calcul des multiplications scalaires $Q = kP$, où Q et P sont 2 points sur une courbe elliptique, et k est un nombre entier positif de longueur n, qui peut être représenté en binaire $k = \sum_{i=0}^{n-1} k_i 2^i$.

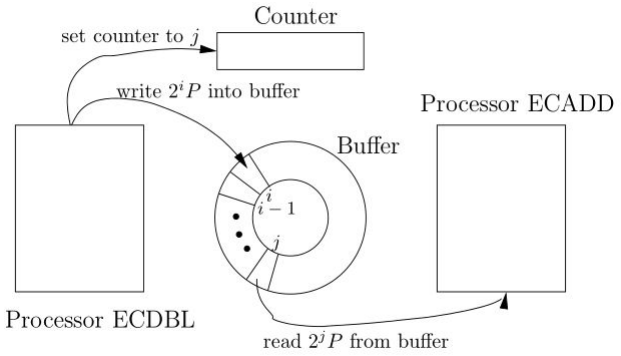


FIGURE 3.4 – Architecture double-processeur pour l'algorithme de calcul parallèle proposé

Le calcul est basé sur l'algorithme 1. Les 2 processeurs partagent une mémoire et un compteur, le premier processeur, noté ECDBL, charge le point P et lit les k_i. Après la lecture de chaque bit, le processeur met à jour le compteur et effectue un double de point $2P$. S'il s'agit d'un bit non nul, il stocke le point $2^i P$ dans la mémoire (voir l'algorithme 8).

Algorithme 8 : Partie ECADD de la multiplication scalaire parallèle

Données : $P, k = \sum_{i=0}^{n-1} k_i 2^i$
Résultat : $2^i P$, stocké dans la mémoire

```
1  i ← 0;
2  Q ← P;
3  tant que i < n faire
4      si k_i = 1 alors
5          si Mémoire est pleine alors
6              Attendre;
7          sinon
8              Stocker Q dans la mémoire;
9          fin
10     fin
11     Q ← 2Q;
12     i ← i + 1;
13     Compteur ← i;
14 fin
```

Le deuxième processeur, noté ECADD, récupère le point $2^i P$ depuis la mémoire et effectue l'addition $Q = Q + 2^i P$. Le calcul est terminé quand tous les bits k_i sont scannés et il

n'y a plus de points en mémoire partagée (voir l'algorithme 9).

Algorithme 9 : Partie ECDBL de la multiplication scalaire parallèle

Données : 2^iP, lu depuis la mémoire
Résultat : $Q = kP$

```
1  Q ← ∞;
2  i ← Compteur;
3  tant que i < n faire
4      si Mémoire n'est pas pleine alors
5          Lire 2^iP depuis la mémoire;
6          Q ← Q + 2^iP;
7          Effacer 2^iP dans la mémoire;
8      fin
9      i ← Compteur;
10 fin
11 retourner Q
```

3.4/ PROTOCOLES CRYPTOGRAPHIQUES BASÉS SUR ECC

Grâce au problème de logarithme discret, plusieurs protocoles cryptographiques ont été proposés. Dans cette section, nous présentons les protocoles principaux (échange de clés, chiffrement et signature numérique) qui existent dans la littérature [55].

3.4.1/ ELGAMAL

Dans [24], ElGamal propose un protocole de chiffrement et de signature numérique basé sur le protocole d'échange de clé Diffie-Hellman [22]. Dans le protocole Diffie-Hellman, A et B veulent partager un secret entre eux, et chacun détient une clé privée appelé respectivement x_A et x_B, p est un grand nombre premier, et α est la racine primitive modulo p (le générateur). A calcule $y_A \equiv \alpha^{x_A} \pmod{p}$ et l'envoie à B, et dans l'autre sens, B calcule $y_B \equiv \alpha^{x_B} \pmod{p}$ et l'envoie à A (voir figure 3.5). Le secret partagé K_{AB} est donc

$$\begin{aligned} K_{AB} &\equiv \alpha^{x_A x_B} \pmod{p} \\ &\equiv y_A^{x_B} \pmod{p} \\ &\equiv y_B^{x_A} \pmod{p} \end{aligned} \tag{3.29}$$

Grâce à la difficulté pour résoudre le problème du logarithme discret, A et B peuvent s'envoyer y_A et y_B sans risque, car il est extrêmement difficile de calculer les valeurs x_A et x_B. Une fois le secret partagé établi entre A et B, ils peuvent sécuriser la communication entre eux avec un algorithme de cryptographie symétrique qui est moins lourd à gérer.

3.4.1.1/ CHIFFREMENT ET DÉCHIFFREMENT

Un algorithme de chiffrement et déchiffrement à clé public est aussi proposé dans [24]. Si A veut envoyer un message m à B où $m \in [0, p-1]$. A choisit d'abord un nombre entier

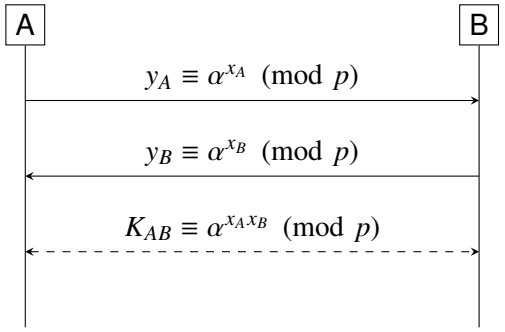


FIGURE 3.5 – Échange de clés Diffie-Hellman

$k \in [0, p-1]$ et on calcule

$$K \equiv y_B^k \pmod{p} \tag{3.30}$$

où $y_B \equiv \alpha^{x_B} \pmod{p}$ est en effet la clé publique de B. Pour chiffrer le message, A calcule

$$c_1 \equiv \alpha^k \pmod{p} \qquad c_2 \equiv Km \pmod{p} \tag{3.31}$$

et envoie c_1 et c_2 à B. Pour déchiffrer le message, B calcule dans un premier temps la valeur de $K \equiv (\alpha^k)^{x_B} \equiv c_1^{x_B} \pmod{p}$, car x_B est la clé privée de B, ensuite B divise c_2 par K pour récupérer le message m. Le déroulement du protocole est illustré graphiquement dans la figure 3.6.

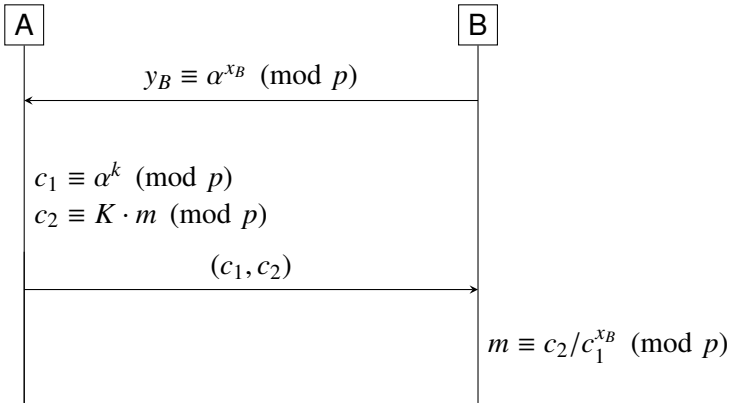


FIGURE 3.6 – Protocole de chiffrement d'Elgamal

Cet algorithme peut aussi être appliquer sur les courbes elliptiques. Supposons que G soit le point générateur sur une courbe définie dans un corps premier fini, notée $E(\mathbb{F}_p)$. A et B disposent chacun d'une clé privée, x_A et x_B avec lesquelles ils peuvent calculer leurs clés publiques P_A et P_B.

$$P_A = x_A.G \quad \text{et} \quad P_B = x_B.G \tag{3.32}$$

Lorsque A veut envoyer un message m à B, il convertit d'abord m en un point, noté M, sur la courbe $E(\mathbb{F}_p$, et prendre un nombre entier aléatoire $k \in [0, p-1]$, ensuite il commence à chiffrer le message en calculant

$$M_1 = k.G \qquad M_2 = M + k.P_B \tag{3.33}$$

Enfin le couple (M_1, M_2) est envoyé à B qui peut déchiffrer le message de la manière suivante.

$$M = M_2 - x_B.M_1 = M + k.P_B - x_B.k.G = M + (k.P_B - k.P_B) \tag{3.34}$$

3.4.1.2/ SIGNATURE NUMÉRIQUE

L'objectif de la signature numérique est de garantir l'intégrité d'un document et d'authentifier l'auteur. En vérifiant la signature, le récepteur du document peut identifier le signataire du document et détecter si le document a été modifié pendant la transmission.

Pour signer un document, le signataire doit trouver une solution qui lui permette de signer le document un utilisant sa clé privée, et les autres personnes peuvent vérifier la signature avec la clé publique du signataire.

Dans la proposition de ElGamal, la signature d'un document $m \in [0, p-1]$ est un couple (r, s) tel que

$$\alpha^m \equiv y^r r^s \pmod{p} \tag{3.35}$$

où $y \equiv \alpha^x \pmod{p}$ est la clé publique du signataire, et x est sa clé privée.

Avant de lancer la procédure de signature, le signataire trouve un nombre entier $k \in [0, p-1]$ tel que $PGCD(k, p-1) = 1$. Ensuite il calcule

$$r \equiv \alpha^k \pmod{p} \tag{3.36}$$

et la formule 3.35 peut être transformée à

$$\alpha^m \equiv \alpha^{xr}\alpha^{ks} \pmod{p} \tag{3.37}$$

à partir de laquelle nous pouvons calculer s en utilisant l'équation

$$m \equiv xr + ks \pmod{p-1} \tag{3.38}$$

Une fois que le document est arrivé chez destinataire, il suffit que le récepteur calcule les deux côtés de l'équation 3.35 pour voir s'ils sont égaux.

Cet algorithme doit être légèrement modifié avant de pouvoir être utilisé avec les courbes elliptiques [81]. Afin de sécuriser la communication entre eux, utilisateurs A et B choisissent la même courbe $E(\mathbb{F}_p)$ dont le point de générateur est G. Avant d'envoyer le message M à B, il faut qu'il soit signé par A. Supposons que la clé privée de A soit x_A, et avec laquelle on peut calculer facilement sa clé publique $P_A = x_A G$. La procédure de signature est constituée de 5 étapes.

1. Choisir d'abord un nombre entier aléatoire $k \in [0, p-1]$.

2. Calculer $R = kG = (x_R, y_R)$ et $r \equiv x_R \pmod{p}$. Si $r = 0$, répéter l'étape précédente.
3. Calculer $e = h(M)$ où h est une fonction de hachage.
4. Calculer $s \equiv k^{-1}(e + rx_A) \pmod{p}$. Si $s = 0$, recommencer depuis l'étape 1.
5. Envoyer (R, s) à utilisateur B.

Après la réception de (R, s), afin de vérifier la signature du document, B calcule $V_1 = sR$ et $V_2 = h(M)G + rP_A$. La signature est valide si $V_1 = V_2$ (voir figure 3.7).

$$\left.\begin{aligned} V_1 &= k^{-1}(e + rx_A)kG = G(h(M) + rx_A) \\ V_2 &= h(M)G + rx_AG = G(h(M) + rx_A) \end{aligned}\right\} \tag{3.39}$$

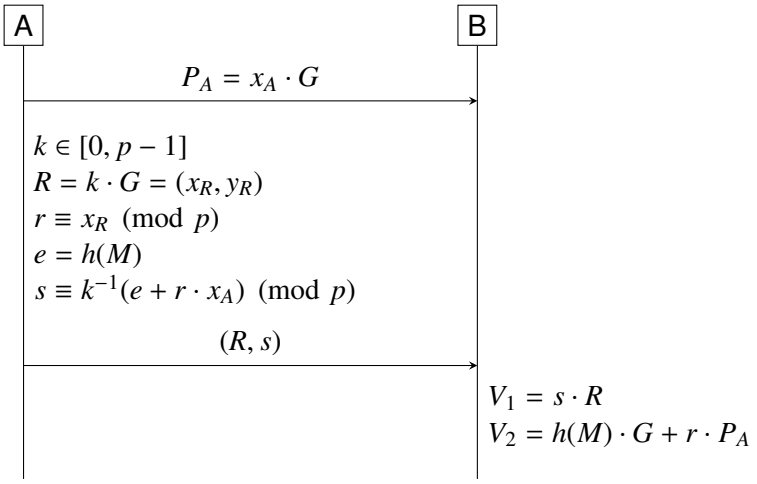


FIGURE 3.7 – Protocole de signature numérique d'Elgamal

3.4.2/ ELLIPTIC CURVE INTEGRATED ENCRYPTION SCHEME (ECIES)

Le protocole d'Elgamal est rarement utilisé directement avec les courbes elliptiques. Avant de chiffrer un message, il faut d'abord le convertir à un point sur la courbe elliptique utilisée. Il y a différentes techniques qui existent, mais la conversion nécessite plus de calcul.

Généralement on utilise les courbes elliptiques pour établir une clé partagée entre les 2 parties d'une conversation [22], ensuite nous pouvons utiliser un algorithme de cryptographie symétrique pour sécuriser la communication entre elles.

Le protocole ECIES est en effet une variante d'Elgamal standardisée. Supposons qu'Alice désire envoyer un message M à Bob d'une manière sécurisée, ils doivent d'abord disposer de toutes les informations suivantes :

– KDF (Key Derivation Function) : Une fonction de dérivation de clé qui permet de générer plusieurs clés à partir d'une valeur secrète de référence.
– MAC (Message Authentication Code) : Code transmis avec les données dans le but d'assurer l'intégrité de ces dernières.
– SYM : Algorithme de chiffrement symétrique.
– $E(\mathbb{F}_p)$: La courbe elliptique utilisée avec le point de générateur G dont $ord_p(G) = n$.

– K_B : La clé publique de Bob $K_B = k_B.G$ où $k_B \in [1, n-1]$ est sa clé privée.

Pour chiffrer le message M, Alice doit effectuer des opérations suivantes :

1. Choisir un nombre entier $k \in [1, n-1]$ et calculer $R = k.G$.
2. Calculer $Z = k.K_B$.
3. Générer les clés $(k_1, k_2) = KDF(abscisse(Z), R)$.
4. Chiffrer le message $C = SYM(k_1, M)$.
5. Générer le code MAC $t = MAC(k_2, C)$.
6. Envoyer (R, C, t) à Bob.

Pour déchiffrer le message (R, C, t), Bob doit effectuer des calculs ci-dessous :

1. Rejeter le message si $R \notin E(\mathbb{F}_p)$.
2. Calculer $Z = k_B.R = k_B.k.G = k.K_B$.
3. Générer les clés $(k_1, k_2) = KDF(abscisse(Z), R)$.
4. Générer le code MAC $t' = MAC(k_2, C)$.
5. Rejeter le message si $t' \neq t$.
6. Déchiffrer le message $M = SYM^{-1}(k_1, C)$.

Dans ce protocole, la valeur critique est k avec laquelle Bob peut calculer $Z = k.K_B$, et générer le couple (k_1, k_2) qui est utilisé pour déchiffrer et authentifier le message. Grâce à la difficulté pour résoudre le problème de logarithme discret, Alice peut envoyer $R = k.G$ sans problème. Une représentation graphique du protocole est illustrée dans la figure 3.8, et une description plus détaillée est donnée dans [9] avec les paramètres recommandés dans [87].

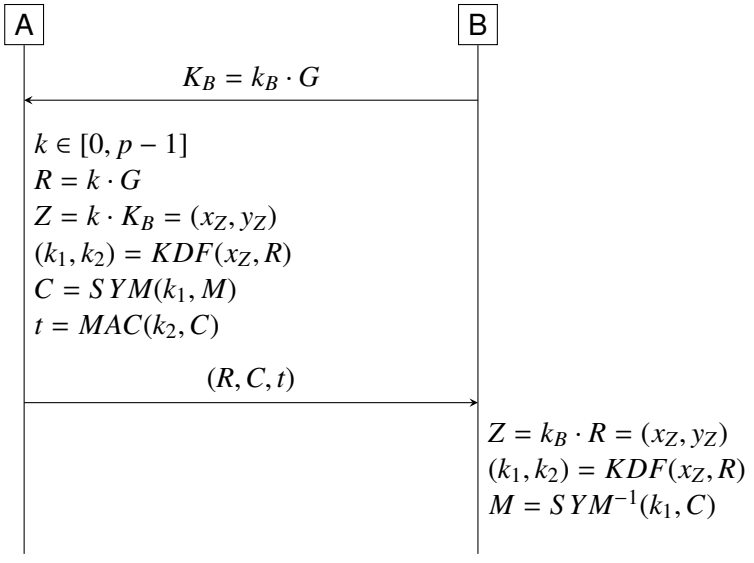


FIGURE 3.8 – Protocole de chiffrement ECIES

3.4.3/ Elliptic Curve Digital Signature Algorithm (ECDSA)

Le protocole ECDSA est proposé par Johnson et al. dans [40], il est une variante de DSA qui utilise les techniques de cryptographie sur les courbes elliptiques. Le protocole DSA signifie Digital Signature Algorithm en Anglais, c'est un algorithme de signature numérique standardisé par le NIST aux États-Unis [20].

Le protocole est basé sur l'idée du protocole de signature d'Elgamal. Nous supposons qu'Alice et Bob utilisent la même courbe elliptique $E(\mathbb{F}_p)$ pour sécuriser la communication entre eux. Nous supposons que la clé publique d'Alice est $K_A = k_A.G$ où k_A est sa clé privée et G est le point de générateur de l'ordre n.

Pour signer un message M, Alice doit suivre les opérations suivantes :

1. Choisir un nombre aléatoire $k \in [1, n-1]$.
2. Calculer $R = k.G$.
3. Calculer $r \equiv abscisse(R) \pmod{n}$. Si $r = 0$, retourner à l'étape 1.
4. Calculer $s \equiv k^{-1}(H(M) + k_A r) \pmod{n}$ où H est une fonction de hachage. Si $s = 0$, retourner à l'étape 1.
5. Envoyer (r, s) à Bob.

Après avoir reçu le message signé, Bob vérifie la signature du message :

1. Vérifier si $K_A \neq \infty$ (point à l'infini) et $K_A \in E(\mathbb{F}_p)$.
2. Vérifier si $n.K_A = \infty$ car $n.K_A = n.k_A.G$ et $ord_p(G) = n$.
3. Vérifier si $(r, s) \in [1, n-1]$.
4. Calculer $R = (H(M)s^{-1} \bmod n)G + (rs^{-1} \bmod n)K_A$ (voir la formule 3.40).
5. Vérifier si $r \equiv abscisse(R) \pmod{n}$.

$$\begin{aligned} R &= (H(M)s^{-1})G + (rs^{-1})K_A \pmod{n} \\ &= (H(M)s^{-1})G + (rs^{-1})k_A G \pmod{n} \\ &= s^{-1}G(H(M) + rk_A) \pmod{n} \\ &= k(H(M) + k_A r)^{-1}G(H(M) + rk_A) \pmod{n} \\ &= kG \end{aligned} \tag{3.40}$$

Le déroulement du protocole est montré dans la figure 3.9.

3.4.4/ Elliptic Curve Menezes Qu Vanstone (ECMQV)

ECMQV est un protocole d'échange de clés authentifié proposé dans [51], car en utilisant Diffie-Hellman, nous ne pouvons pas assurer l'authentification des participants

Nous supposons que $R(x_R, y_R)$ est un point sur une courbe elliptique, et P est le point de générateur dont $ord(P) = n$. Nous calculons $\bar{R} = (x_R \bmod 2^L) + 2^L$ où $L = \lceil \frac{\lfloor \log_2 n \rfloor + 1}{2} \rceil$, alors $\bar{R}$ est en effet les premier L bits de la coordonnée abscisse de R. En outre, (q_A, Q_A) et (q_B, Q_B) sont respectivement la clé privée et la clé publique de Alice et Bob.

La procédure d'échange de clés commence par une génération de clés temporaire chez Alice :

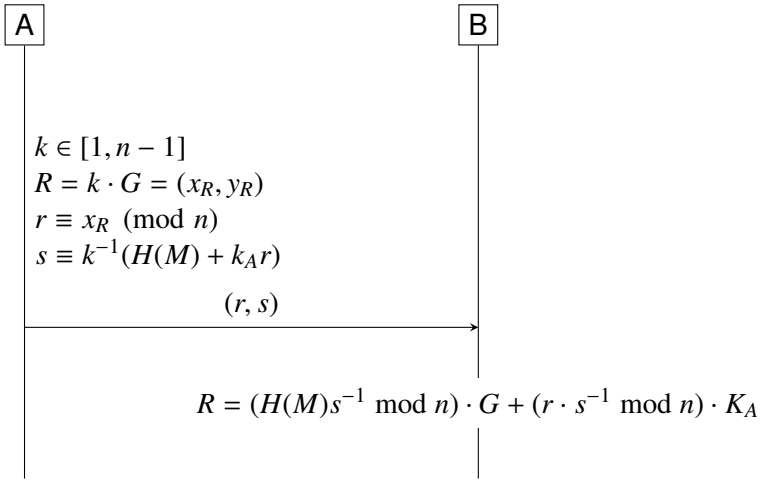


FIGURE 3.9 – Protocole de signature numérique ECDSA

– Choisir en nombre entier $k_A \in [1, n-1]$ et calculer $R_A = k_A.P$.
– Envoyer R_A à Bob.

Après la réception de R_A, Bob effectue des opérations suivantes :

– Choisir un nombre entier aléatoire $k_B \in [1, n-1]$ et calculer $R_B = k_B.P$.
– Calculer $s_B \equiv (k_B + \bar{R}_B q_B) \pmod{n}$ (Bob signe sa clé temporaire).
– Calculer $Z = s_B(R_A + \bar{R}_A Q_A)$.
– Génération de clés $(k_1, k_2) = KDF(abscisse(Z))$ où KDF est une fonction de dérivation de clés.
– Génération de MAC $t_B = MAC(k_1, 2, B, A, R_B, R_A)$ où 2 est un nonce cryptographique.
– Envoyer (R_B, t_B) à Alice.

Alice reçoit (R_B, t_B) et vérifie l'identité de Bob en faisant :

– Calculer $s_A \equiv (k_A + \bar{R}_A q_A) \pmod{n}$ (Alice signe sa clé temporaire).
– Calculer $Z' = s_A(R_B + \bar{R}_B Q_B)$ (voir la formule 3.41).
– Génération de clés $(k_1, k_2) = KDF(abscisse(Z'))$.
– Vérifier si $t_B = MAC(k_1, 2, B, A, R_B, R_A)$.
– Génération de MAC $t_A = MAC(k_1, 3, A, B, R_A, R_B)$.
– Envoyer t_A à Bob.

Bob vérifie l'identité d'Alice :

– Vérifier si $t_A = MAC(k_1, 3, A, B, R_A, R_B)$.

$$\begin{aligned} Z &= s_B(R_A + \bar{R}_A Q_A) \\ &= (k_B + \bar{R}_B q_B)(R_A + \bar{R}_A Q_A) \\ &= (k_B + \bar{R}_B q_B)(k_A + \bar{R}_A q_A)P \end{aligned} \qquad \begin{aligned} Z' &= s_A(R_B + \bar{R}_B Q_B) \\ &= (k_A + \bar{R}_A q_A)(R_B + \bar{R}_B Q_B) \\ &= (k_A + \bar{R}_A q_A)(k_B + \bar{R}_B q_B)P \end{aligned} \tag{3.41}$$

Le secret partagé k_2 entre les 2 participants de la communication, et contrairement au protocole de Diffie-Hellman, la clé de session n'est jamais la même. Seulement Alice et Bob peuvent calculer Z, et ils utilisent leurs clés temporaires k_A et k_B pour prouver leurs identités. Le schéma qui représente l'établissement de la clé de session est donné dans

la figure 3.10.

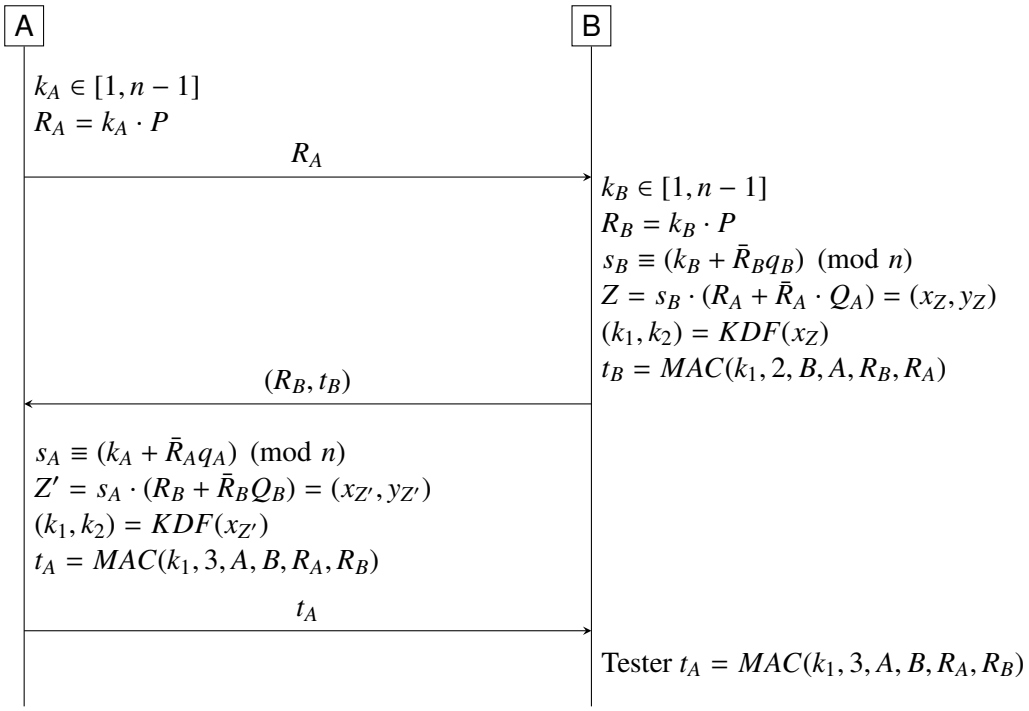


FIGURE 3.10 – Protocole d'échange de clés ECMQV

3.4.5/ ELLIPTIC CURVE MASSEY-OMURA (EC MASSEY-OMURA)

En cryptographie, le protocole à trois passes, permet à Alice d'envoyer un message chiffré à Bob sans avoir besoin d'échanger ou de distribuer des clés. Le premier protocole à trois passes était celui développé par Shamir en 1980. Il est aussi appelé Shamir No-Key Protocol, car l'émetteur et le récepteur n'ont pas besoin d'échanger des clés. Le protocole nécessite néanmoins deux clés privées pour crypter et décrypter les messages.

James Masey et Jim K. Omura ont proposé une amélioration du protocole de Shamir en 1982 [67]. La méthode de Massey-Omura utilise l'exponentiation dans le corps de Galois $GF(2^n)$. Le message chiffré est M^e et le message déchiffré est M^d. Les calculs sont effectués dans le corps de Galois. Quelque soit le nombre entier e utilisé pour chiffrer le message avec $0 < e < 2^n - 1$ et $PGCD(e, 2^b - 1) = 1$, le nombre entier d permettant de déchiffrer le message est tel que $de \equiv 1 \pmod{2^n - 1}$. Comme le groupe multiplicatif associé au corps de Galois $GF(2^n)$ est de l'ordre $2^n - 1$, le théorème de Lagrange implique que $m^{de} = m$ pour tout m dans $GF(2^n)^*$.

La Théorème de Lagrange montre que pour un groupe G fini et pour tout sous-groupe H de G, l'ordre de H divise l'ordre de G. Dans notre cas, le groupe considéré est le groupe abélien associé à notre courbe elliptique E. Soit N le nombre de points sur la courbe $E(\mathbb{F}_p)$, ce nombre est public. Alice et Bob choisissent leur clé secrète : deux nombres entiers aléatoires e_A et e_B tel que $PGCD(e_A, n) = 1$ et $PGCD(e_B, n) = 1$. Ils calculent $d_A \equiv e_A^{-1} \pmod{N}$ et $d_B \equiv e_B^{-1} \pmod{N}$ en utilisant l'algorithme d'Euclide. Si Alice veut envoyer le message M à Bob, elle envoie d'abord $e_A M$. Toutefois, Bob ne peut pas le déchiffrer, il renvoie $e_B e_A M$) Alice. Alice renvoie alors $d_A e_B e_A M$ à Bob, et comme $d_A e_A \equiv 1 \pmod{N}$,

nous avons $d_a e_B e_A M = e_B M$ et donc Bob peut déchiffrer le message.

3.5/ COMPARAISON DE PERFORMANCE ENTRE ECC ET RSA

RSA fait partie des premiers cryptosystèmes asymétriques qui sont largement appliqués, et aujourd'hui, il est toujours considéré comme le cryptosystème asymétrique le plus utilisé, notamment pour échanger des données confidentielles sur internet. Il est proposé en 1977 par Ron Rivest, Adi Shamir et Leonard Adleman [89]. L'algorithme est basé sur l'hypothèse qu'il est extrêmement difficile d'effectuer la factorisation d'un nombre entier en facteurs premiers. Certes, la factorisation n'est pas la seule méthode pour casser RSA, mais pour l'instant, il n'existe pas d'autre attaque qui soit suffisamment efficace.

RSA est proposé avec 3 algorithmes qui sont conçus respectivement pour la génération de clés, la signature numérique et la vérification de signature.

3.5.1/ GÉNÉRATION DE CLÉS

Avec RSA, la clé publique et la clé privée du système peuvent être générées de la manière suivante :

- Choisir 2 nombres premiers aléatoires p et q ayant approximativement la même longueur en nombre de bits.
- Calculer $n = p \cdot q$.
- Calculer $\phi(n) = (p-1) \cdot (q-1)$.
- Choisir un nombre entier $e \in]1, \phi(n)[$ tel que $gcd(e, \phi(n)) = 1$.
- Calculer d tel que $e \cdot d \equiv 1 \pmod{n}$.
- (n, e) est la clé publique et d est la clé privée.

3.5.2/ SIGNATURE NUMÉRIQUE

La signature numérique permet au destinataire d'assurer l'intégrité du message et d'authentifier son auteur. Afin de signer le message m avec RSA, il suffit de chiffrer le message avec la clé privée.

$$s \equiv hash(m)^d \pmod{n} \tag{3.42}$$

$hash$ est une fonction de hachage, et généralement nous utilisons la fonction SHA-1. Une fois le message est signé, le signataire envoie le message m avec sa signature au destinataire.

3.5.3/ VÉRIFICATION DE SIGNATURE

Pour vérifier la signature reçue, le destinataire doit déchiffrer le message avec la clé publique du signataire.

$$h \equiv s^e \equiv (hash(m)^d)^e \equiv hash(m) \pmod{n} \tag{3.43}$$

Pour authentifier l'auteur du message, le destinataire applique la même fonction de hachage au m et tester si le résultat est égal à h.

3.5.4/ COMPARAISON DE PERFORMANCE

Afin d'évaluer et de comparer la performance de RSA et ECC, 2 implémentations utilisant respectivement ces 2 cryptosystèmes sont testés dans [39]. Comme nous avons présenté au début du chapitre, ECC peut avoir le même niveau de sécurité que RSA avec une clé beaucoup plus courte. Les longueurs de clé utilisées sont suggérées dans [54] (voir le tableau 3.3), et les longueurs qui se situent dans la même colonne sont censées pouvoir fournir le même niveau de robustesse. D'ailleurs, l'algorithme de signature utilisé pour ECC est ECDSA, une variante de DSA conçue pour les courbes elliptiques, et un texte de longueur 100 Ko est utilisé pour tester la signature.

Algorithme	Longueur de clé (bit)				
ECC	163	233	283	409	571
RSA	1024	2240	3072	7680	15360

TABLE 3.3 – Longueur de clé en bit

Les résultats de test sont donnés dans les tableaux 3.4, 3.5 et 3.6. Nous pouvons constater que pour atteindre le même niveau de sécurité, premièrement, la consommation de mémoire est beaucoup moins importante avec ECC, car nous utilisons des clés plus courtes. Deuxièmement, les calculs de la génération de clé et de la signature de message sont plus rapide avec ECC. La vérification de signature est plus rapide avec RSA, car il suffit d'effectuer une exponentiation modulaire.

Longueur de clé		Temps de calcul (s)	
ECC	RSA	ECC	RSA
163	1024	0.08	0.16
233	2240	0.18	7.47
283	3072	0.27	9.80
409	7680	0.64	133.90
571	15360	1.44	679.06

TABLE 3.4 – Temps d'exécution pour la génération de clés

Le même type de test a été effectué dans [32, 43], les résultats de test ont aussi prouvé que ECC était plus avantageux en terme de consommation de mémoire et vitesse de calcul. *C'est la raison principale pour laquelle ECC devient de plus en plus le choix préféré pour les systèmes embarqués qui disposent d'une mémoire et d'une puissance de calcul très limitée*.

Actuellement RSA demeure toujours le cryptosystème asymétrique le plus largement utilisé, car il est sorti beaucoup plus tôt par rapport à ECC. RSA est publié en 1978 [89] et standardisé en 1993 [49], tandis que ECC est proposé dans les années 80 [44, 69] et standardisé vers la fin des années 90. Les 2 cryptosystèmes sont initialement protégés

Longueur de clé		Temps de calcul (s)	
ECC	RSA	ECC	RSA
163	1024	0.15	0.01
233	2240	0.34	0.15
283	3072	0.59	0.21
409	7680	1.18	1.53
571	15360	3.07	9.20

TABLE 3.5 – Temps d'exécution pour la signature numérique

Longueur de clé		Temps de calcul (s)	
ECC	RSA	ECC	RSA
163	1024	0.23	0.01
233	2240	0.51	0.01
283	3072	0.86	0.01
409	7680	1.80	0.01
571	15360	4.53	0.03

TABLE 3.6 – Temps d'exécution pour la vérification de signature

par des brevets. Du fait que le brevet de RSA a expiré depuis 2000, il peut être utilisé librement par tout le monde, mais celui de ECC est toujours valable [14].

Une autre raison qui limite l'utilisation de ECC est que mathématiquement RSA est relativement plus simple. De nombreuse spécifications de ECC existent [74], mais les implémentations sont souvent incomplètes, c'est-à-dire que seulement quelques courbes décrites dans les spécifications sont implémentées.

3.6/ ATTAQUES DES CRYPTOSYSTÈMES EMBARQUÉS BASÉS SUR ECC

Du point de vue mathématique, les cryptosystèmes embarqués basés sur les courbes elliptiques sont censés être suffisamment sûrs. Cependant, ils peuvent toujours faire l'objet d'attaques matérielles. Il existe 3 types d'attaques selon leur nature : non-invasives, semi-invasive et invasive [28].

Une attaque non-invasive peut être, soit une observation de certains caractéristiques d'un cryptosystème durant la manipulation de sa clé secrète, soit une injection de fautes pour perturber les calculs. Le premier cas s'appelle aussi l'attaque par canaux cachés.

Le premier document officiel relatant une utilisation d'un canal caché date de 1956 [110], le service de renseignement britannique a cassé une machine de chiffrement de l'ambassade égyptienne en observant les « clic » produits par des rotors (voir la figure 3.11).

Pour les systèmes embarqués modernes, les différences de tension engendrées par les transistors durant un cycle d'horloge dépendent des instructions exécutées et des valeurs utilisées. L'examen de la consommation d'énergie permet de trouver des informations sur la clé secrète [45] (voir la figure 3.12). Une autre possibilité est d'analyser les signaux engendrés par le rayonnement électromagnétique pendant la circulation d'un courant.

FIGURE 3.11 – Machine de chiffrement à rotor

Les résultats permettent aussi de récupérer les informations sur les opérations effectuées par le cryptosystème, ainsi que les variables manipulées [104].

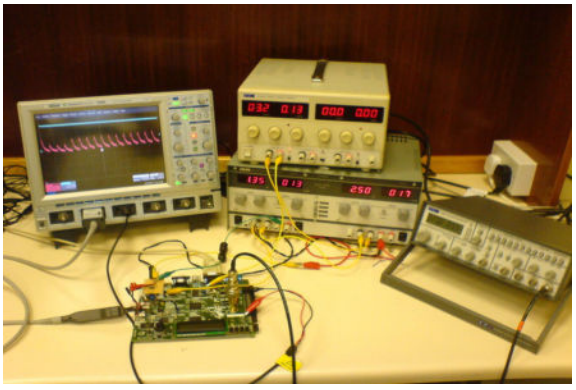


FIGURE 3.12 – Attaque par analyse de la consommation électrique

Le deuxième type d'attaque non-invasive est d'alterer le fonctionnement du cryptosystème en modifiant l'environnement, e.g. modification de température, injection des pics de tension etc. Ce genre d'attaque est censé être paramétrable pour ne pas endommager le circuit, et lorsque l'attaque cesse, le circuit peut retrouver son état initial.

Les attaques semi-invasives sont similaires aux attaques non-invasives. Nous observons les caractéristiques d'un cryptosystème, sauf qu'avec l'attaque semi-invasive, nous devons décapsuler le package pour être plus près possible du circuit. Ainsi, l'attaquant peut altérer le fonctionnement du cryptosystème en créant des circuits additifs ou émettant des ondes électromagnétiques pour perturber temporairement le circuit [94].

Contrairement à l'attaque non-invasive et l'attaque semi-invasive, avec l'attaque invasive, la modification et la perturbation du cryptosystème sont permanentes.

3.7/ CONCLUSION

Dans ce chapitre, nous avons présenté la cryptographie sur les courbes elliptiques, ainsi que l'ensemble de termes mathématiques indispensables pour comprendre son fonctionnement. En faisant des comparaisons de performance avec l'autre cryptosystème largement utilisé, RSA, nous pouvons remarquer que ECC est un cryptosystème plus attractif, notamment pour les systèmes embarqués, comme les capteurs sans fil.

Afin d'atteindre un niveau de sécurité suffisant avec RSA, il faut utiliser des clés de longueurs 1024 minimum, et la mémoire du système risque d'être saturée facilement durant les calculs. Cependant avec ECC, une clé de longueur 160 est déjà capable de fournir le même niveau de sécurité. Une autre raison pour laquelle ECC est préférable pour les systèmes embarqués est qu'il existe beaucoup de méthodes mathématiques qui nous permettent d'accélérer les calculs sur les courbes.

Les capteurs sans fil ne détiennent qu'une mémoire et une puissance de calcul limitée, et les calculs cryptographiques avec ECC restent toujours très lourds, malgré ses avantages par rapport à RSA. Dans une application de réseaux de capteurs sans fil, les capteurs fonctionnent rarement individuellement, car ils sont censés être déployés en masse et coopérer entre eux. Dans le chapitre suivant, nous présentons notre méthode qui nous permet d'accélérer les calculs sur les courbes elliptiques en parallélisant les tâches de calcul.

4

MULTIPLICATION SCALAIRE PARALLÈLE DANS LES RÉSEAUX DE CAPTEURS

Sommaire

4.1/ INTRODUCTION

Dans le chapitre 3, nous avons présenté la cryptographie sur les courbes elliptiques, ainsi que l'ensemble de techniques qui nous permettent d'accélérer le calcul des multiplications scalaires, qui est considérée comme l'opération la plus importante et coûteuse sur les courbes elliptiques. Après une étude des protocoles cryptographiques qui sont

basés sur ECC, nous pouvons constater que la sécurité de ECC dépend principalement de la difficulté pour résoudre le problème du logarithme discret.

Un des avantages de ECC par rapport à RSA est la rapidité de calcul, et beaucoup de méthodes mathématiques sont proposées dans la littérature pour améliorer sa performance au niveau de la vitesse de calcul. D'ailleurs, une autre caractéristique de ECC très importante, notamment pour les plate-formes embarqués disposant d'une mémoire très limitée, est qu'afin d'atteindre le même niveau de robustesse que RSA, il nous suffit d'utiliser une clé beaucoup plus courte.

Un réseau de capteurs sans fil est constitué d'un grand nombre de capteurs sans fil, appelés aussi des *nœuds*, qui sont très limités en termes de puissance de calcul et de mémoire disponible. Ces dispositifs ne sont pas censés être capables de gérer des calculs très complexes, dans les réseaux de capteurs sans fil, les nœuds sont déployés en masse, car il faut qu'ils coopèrent étroitement entre eux pour accomplir un objectif collectif.

Revenons sur le sujet du calcul cryptographique. Donc ECC est un choix optimal pour les réseaux de capteurs. C'est un cryptosystème asymétrique qui donne la possibilité de gérer plus facilement les clés, car les clés publiques peuvent être distribuées librement entre les nœuds sans compromettre la sécurité du réseau. En outre, idéalement les nœuds sont censés être déployés aléatoirement dans une zone, si nous utilisons simplement un cryptosystème symétrique, la solution la plus naïve pour gérer des clés sera de générer et de stocker une clé secrète pour chaque paire de nœuds, car nous ne pouvons pas savoir à priori les nœuds voisins qui vont se trouver à la proximité d'un nœud spécifique.

Un inconvénient principal des cryptosystèmes asymétriques est qu'ils sont souvent plus coûteux en calcul. Généralement les processeurs modernes sont capables de gérer aisément ce genre de calculs, mais la plupart des capteurs sans fil sont équipés des microcontrôleurs qui tournent à une fréquence très faible [72, 19, 18]. Du fait de leur faiblesse en puissance, il est déconseillé d'appliquer la cryptographie sur les courbes elliptiques, telle quelle, dans les réseaux de capteurs malgré de nombreuses méthodes mathématiques existantes qui sont conçues pour l'optimisation de sa performance. Les nœuds risquent d'être figés durant des calculs à cause d'un épuisement de mémoire ou d'une surcharge de processeur.

Afin d'équilibrer la charge entre des nœuds et d'accélérer les calculs sur les courbes elliptiques, dans ce chapitre, nous proposons de paralléliser le calcul des multiplications scalaires entre plusieurs nœuds d'un cluster [36, 35]. En prenant en compte le fonctionnement des réseaux de capteurs, nous avons décidé de décomposer le calcul des multiplications scalaires, l'opération la plus importante sur les courbes elliptiques, en plusieurs tâches complètement indépendantes qui peuvent être traitées simultanément par les membres de cluster. Une fois terminés, les résultats seront renvoyés au cluster-head qui les combinera ensemble pour obtenir le résultat final. Ce dernier peut être ensuite réutilisé dans de différentes phases de calcul des protocoles cryptographiques. *Nos travaux de recherche se focalisent principalement sur la parallélisation de la multiplication scalaire*.

Dans les sections suivantes, nous allons présenter les bases théoriques de la multiplication scalaire parallèle dans les réseaux de capteurs, ainsi que le protocole utilisé. Pour tester la performance de notre méthode, nous l'avons implémentée sur PC et sur les nœuds Telosb [29, 79, 18], les résultats d'expérimentation ont montré que le calcul paral-

lèle peut permettre une accélération de calcul très importante.

4.2/ CALCUL PARALLÈLE

En informatique, le calcul parallèle consiste à utiliser des architectures permettant de traiter des informations simultanément. Cette technique est initialement mise en place pour gérer des tâches complexes et volumineuses qui doivent être réalisées dans un délai le plus court possible. D'un point de vue de la conception de processeurs, pour accélérer le traitement d'informations, la solution la plus simple est d'augmenter sa fréquence d'horloge, c'est-à-dire le nombre d'opérations qu'il peut effectuer en une seconde. Cependant ce dernier est devenu le goulot d'étranglement à cause des limites physiques, notamment dans les domaines du Calcul Haute Performance[1]. Récemment avec l'arrivée des processeurs multi-cœurs qui sont capables de traiter plusieurs instructions en même temps, le calcul parallèle est devenu un paradigme dominant pour tous les ordinateurs.

4.2.1/ TAXONOMIE

Selon la taxonomie de Michael J. Flynn [27], les programmes et ordinateurs peuvent être classés 4 catégories suivantes (voir le tableau 4.1) :

	Un seule instruction	Plusieurs instructions
Donnée unique	SISD	MISD
Plusieurs données	SIMD	MIMD

TABLE 4.1 – Taxonomie de Flynn

– SISD (Single Instruction, Single Data) : C'est l'architecture la plus simple, chaque fois le système ne traite qu'une seule donnée, les instructions sont exécutées l'une après l'autre, d'une manière séquentielle.
– SIMD (Single Instruction, Multiple Data) : C'est le cas où les différentes données sont envoyées aux processeurs qui les traitent en répétant les mêmes opérations.
– MISD (Multiple Instruction, Single Data) : Plusieurs opérations sont effectuées sur la même donnée, cette architecture est parfois utilisée pour la tolérance aux pannes.
– MIMD (Multiple Instruction, Multiple Data) : Plusieurs données sont traitées par des processeurs différents, les systèmes distribués sont généralement considérés comme des architectures MIMD.

2 types de calcul parallèle sont montrés dans cette classification, la parallélisation d'instructions, où plusieurs instructions différentes sont exécutées simultanément par de différents processeurs (MIMD). Le deuxième est la parallélisation de données, où les mêmes opérations sont effectuées sur de différentes données (SIMD).

En fonction de ce que nous devons paralléliser durant les calculs, il est aussi possible de classifier les calculs parallèles en 3 catégories :

– Parallélisation de mots : C'est un type de calcul parallèle basé sur l'augmentation de la longueur du mot des processeurs. Par exemple, un processeur de 8-bit n'est pas capable de gérer des données de 16-bit, pour additionner 2 nombres entiers de 16 bits, le processeur doit additionner d'abord les 8 bits de poids faibles, ensuite additionner les 8

bits de poids forts avec la retenue. Il faut donc 2 opérations pour effectuer une addition, mais un processeur de 16-bit peut terminer le calcul en une seule opération. Une autre solution est d'avoir 2 processeurs de 8-bit reliés, et additionnent simultanément les bits de poids faibles et de poids forts.
– Parallélisation de données : Une grande quantité de données est découpée et distribuée à plusieurs processeurs exécutant les mêmes instructions. Il s'agit d'un calcul parallèle du type SIMD. Par exemple nous disposons d'une base de données contenant d lignes (enregistrements), et de 2 processeurs qui exécutent exactement le même programme. Nous supposons que chaque ligne peut être traitée indépendamment, nous pouvons donc envoyer les $\frac{d}{2}$ premières lignes au premier processeur, et les $\frac{d}{2}$ lignes restantes au deuxième.
– Parallélisation de tâches : Contrairement à la parallélisation des données, au lieu de distribuer des portions de données, la parallélisation des tâches distribue carrément des procédures ou des threads aux différents processeurs qui les exécutent parallèlement.

4.2.2/ ACCÈS AUX DONNÉES

Dans le cadre de calcul parallèle, la mémoire contenant les données qui circulent entre les processeurs peut être soit partagée, soit distribuée.

Dans le premier cas, les processeurs sont censés pouvoir accéder aux données avec la même latence et bande passante. Tous les processeurs participant aux calculs parallèles peuvent y déposer et lire des données, et c'est une configuration idéale qui permet au système d'améliorer considérablement ses performances (voir figure 4.1). Car généralement la mémoire partagée peut offrir un accès aux données plus rapide et fiable, la synchronisation entre les processeurs est relativement plus facile à réaliser.

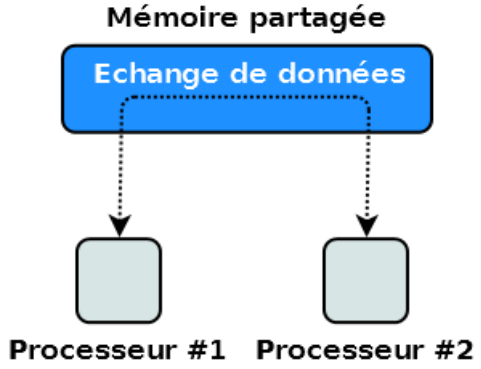


FIGURE 4.1 – Accès aux données avec la mémoire partagée

En revanche, dans le cas de mémoire distribuée, chaque processeur possède sa propre mémoire locale, et durant des calculs, seulement des données locales sont accessibles. Si un processeur a besoin des données qui se situent chez un autre processeur, il faudra qu'il communique avec ce dernier pour les récupérer (voir figure 4.2).

Dans une architecture utilisant la mémoire distribuée, les processeurs doivent souvent communiquer entre eux pour échanger des informations, et un tel accès aux données peut engendrer une baisse de performance importante. D'ailleurs, avant de pouvoir demander des données aux autres, il faut que le processeur connaître la localisation des

données dont il a besoin, car parfois il est possible que les données en question soient hébergées sur des processeurs différents, et la diffusion des requêtes peut aussi ralentir le calcul.

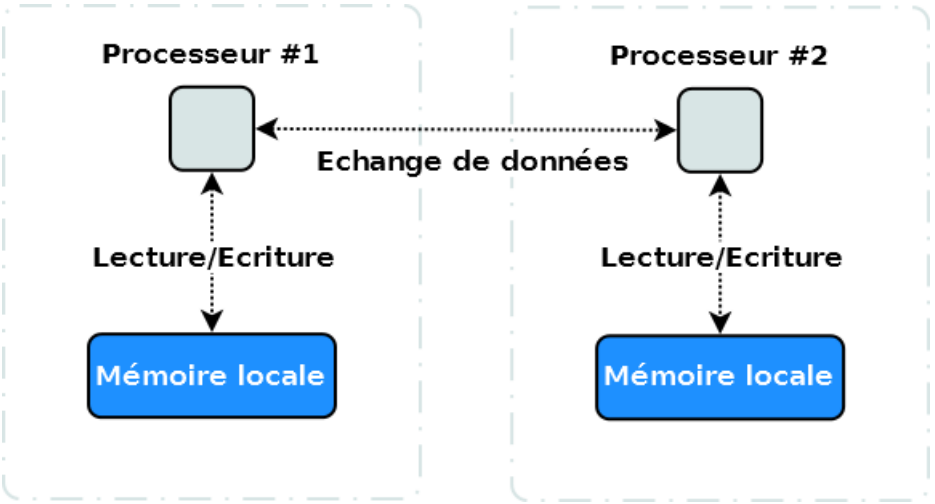


FIGURE 4.2 – Accès aux données avec la mémoire distribuée

4.2.3/ SYNCHRONISATION

Des processus peuvent s'exécuter sur des processeurs différents, lorsqu'il existe des interdépendances sur les données partagées entre des processus, des problèmes de synchronisation peuvent donc être relevés. Lorsqu'un processus effectue une mise à jour des données en mémoire, l'objectif de la synchronisation est de la rendre visible aux autres processus.

Par exemple, le processus p_1 a besoin de la valeur v fournie par le processus p_2 pour continuer son calcul, et en même temps, p_2 veut effectuer une mis à jour la valeur v en mémoire. Pour que les 2 opérations soient synchronisées, il faut que l'accès à la valeur soit *atomique*, c'est-à-dire que la lecture de v et la mise à jour de v ne doivent pas se chevaucher. Soit p_1 lit v après sa mise à jour, soit p_2 met v à jour après la lecture, mais jamais les 2 en même temps (voir figure 4.3).

Chaque opération peut correspondre à plusieurs instructions du processeur, si les instructions des 2 opérations sont exécutées simultanément sur la même donnée, les processus risquent de rencontrer des erreurs imprévisibles. Ainsi, lorsque p_1 détecte que le bloc mémoire contenant v est momentanément occupées par p_2, il est impératif que p_1 attend jusqu'à ce que p_2 finisse son opération courante.

Le mécanisme de gestion de synchronisation à mettre en place peut être différent en fonction du type de mémoire utilisée. Pour une architecture utilisant une mémoire partagée, l'accès à la mémoire est plus facile à surveiller et contrôler, et nous pouvons avoir plus aisément un accès aux données bien synchronisé et cohérent. Cependant, si l'architecture utilise la mémoire distribuée, les processeurs devront donc échanger des messages entre eux pour demander des données.

En faisant la comparaison ci-dessus, il est évident que le calcul parallèle utilisant la mémoire partagée est plus avantageux en termes de performance, car il permet au système

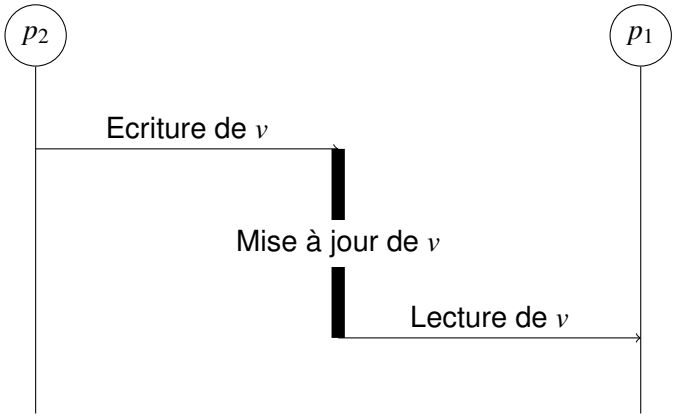


FIGURE 4.3 – Synchronisation d'accès aux données

d'avoir un accès aux données plus efficace et cohérent.

4.2.4/ TOLÉRANCE AUX PANNES

La tolérance aux pannes est en effet un ensemble de techniques qui permettent à un système de continuer à fonctionner en cas de panne. Au lieu de tomber complètement en panne, le composant peut continuer à délivrer, éventuellement de manière réduite, les services initialement prévus [84].

Cette technique n'est pas utilisée uniquement en informatique, mais aussi dans beaucoup d'autres domaines industriels. Prenons un exemple de l'industrie automobile, les fabricants installent souvent 2 capteurs de pédale, dans le cas où la pédale d'accélérateur soit bloqué (enfoncée), le conducteur peut toujours arrêter la voiture en appuyant sur la pédale de frein. Cependant certains fabricants, pour une raison du coût de fabrication, n'installent qu'un seul capteur, lorsque le capteur est tombé en panne, le conducteur est obligé de trouver d'autres moyens pour arrêter la voiture, car la pédale de frein ne peut plus fonctionner non plus.

Idem pour le calcul parallèle, lorsque nous distribuons les tâches aux processeurs, nous n'avons aucune garantie que les processeurs peuvent forcément mener les calculs jusqu'au bout, et que les résultats renvoyés sont toujours corrects. Différents types de pannes pourraient survenir durant le traitement des tâches, coupure de courant, panne matérielle, bogue de programme etc. Particulièrement les architectures distribuées dans lesquelles les processeurs doivent échanger des données pour effectuer les calculs, si des mécanismes de tolérance aux pannes ne sont pas présents, il suffit d'avoir une simple défaillance de réseaux pour interrompre le calcul. D'ailleurs, il faut que le système puisse non seulement détecter des problèmes, mais aussi se restaurer en cas de pannes sévères.

Pour les plate-formes mobiles et embarquées, les techniques de tolérance aux pannes sont dispensables, car pour minimiser la consommation d'énergie et étendre la durée de vie des réseaux, les nœuds utilisent souvent des antennes radio de faible puissance, qui utilisent les ondes comme le médium de transmission. Ce genre de connexion souffre souvent de l'interférence radio et de conflit d'accès au médium qui rendent la communication moins fiable, et les expérimentations ont montré que lorsque le niveau de batterie

d'un nœud est très faible, son antenne risque d'envoyer des données erronées [12].

Généralement il existe 2 approches pour implémenter les mécanismes de tolérance aux pannes. La première est d'utiliser des composants qui sont tolérants aux pannes. En cas d'anomalie, le composant peut continuer à fonctionner en ignorant les dysfonctionnements. le système reste toujours opérationnel car l'opération courante n'est pas interrompue malgré la présence des pannes. Cette méthode est déconseillée pour des composants importants, car nous mettons en place des techniques de tolérance aux pannes pour assurer le *bon fonctionnement* du système, mais pas simplement la continuité des processus.

La deuxième méthode est la redondance, plusieurs composants sont utilisés pour une seule tâche, lorsque le premier composant tombe en panne, un deuxième composant prend automatiquement le relais et continue à traiter la tâche (voir figure 4.4).

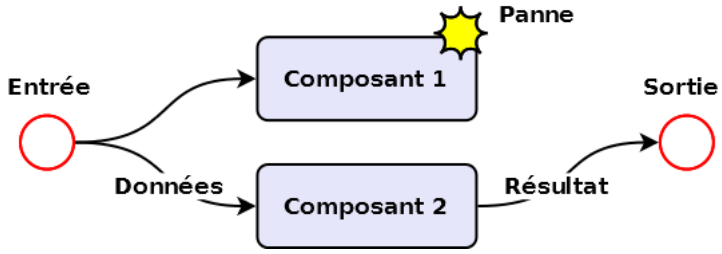


FIGURE 4.4 – Tolérance aux pannes avec la redondance de composants

La redondance peut aussi être utilisée pour la détection de fautes et la correction des données erronées. Prenons l'exemple des réseaux de capteurs sans fil, les données risquent d'être perdues ou corrompues pendant la transmission, lorsqu'un nœud renvoie un résultat de calcul, nous ne pouvons pas tester son intégrité, car généralement le calcul de hachage est très coûteux pour les nœuds. Néanmoins, et nous ne pouvons pas tester si le résultat de calcul est correct non plus, car nous n'en avons qu'une seule copie. Cependant si plusieurs nœuds sont mis en place pour le même calcul, nous pouvons effectuer une vérification en faisant une simple comparaison. Si les résultats ne sont pas identiques, seul la valeur la plus présente est retenue (voir figure 4.5).

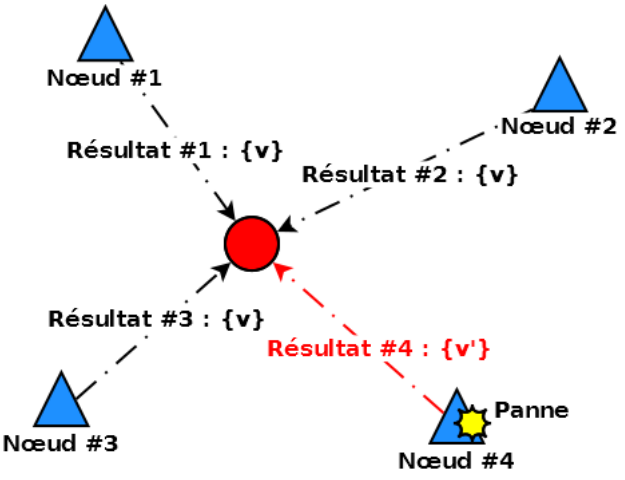


FIGURE 4.5 – Tolérance aux pannes avec la redondance de résultats

Un inconvénient de la redondance est l'augmentation de coût. Pour que le système soit

résistant aux différentes pannes, beaucoup de composants redondants doivent être mis en place. C'est la raison principale pour laquelle la redondance est appliquée seulement aux composants ayant une importance cruciale.

4.3/ CALCUL PARALLÈLE DANS LES RÉSEAUX DE CAPTEURS

Généralement, un réseau de capteurs est constitué d'un grand nombre de nœuds ayant des ressources (puissance de calcul, mémoire, énergie) très limitées, et reliés entre eux via une connexion sans fil de faible puissance. Certaines caractéristiques d'un tel réseau doivent être soulignées avant d'étudier la faisabilité du calcul parallèle.

4.3.1/ ACCÈS AUX DONNÉES

Comme nous avons déjà présenté, les réseaux de capteurs sont constitués des nœuds qui sont physiquement séparés, c'est-à-dire qu'il n'existe pas de mémoires partagés entre les nœuds. Si nous voulons appliquer le calcul parallèle dans les réseaux de capteurs, il faudra que chaque nœud ait en sa mémoire locale toutes les données nécessaires pour la tâche qui lui est assignée. En outre, il faut que ces données soient distribuées aux nœuds participant au calcul parallèle au moment du lancement du calcul, toute échange ultérieure ou distribution séquentielle de données sont très coûteuses en termes d'énergie. Donc les types de parallélisation possibles dans les réseaux de capteurs sont SISD et MISD.

4.3.2/ CONSOMMATION D'ÉNERGIE

Les nœuds sont conçus pour être déployés en masse dans des zones hostiles où l'intervention humaine n'est pas toujours possible. Ils sont souvent alimentés par des piles, et une fois déployés, nous ne sommes pas censés retrouver et maintenir les nœuds dont les piles sont épuisées.

Selon les résultats expérimentaux [112, 2], la plupart d'énergie d'un nœud est consommée pendant la communication de radio, la consommation du processeur est presque négligeable par rapport à celle de l'antenne. Ainsi, tout protocole doit être soigneusement élaboré pour minimiser la communication sans fil.

Idem pour le calcul parallèle, qui fait face au problème de consommation d'énergie depuis son invention. Afin d'alimenter une architecture effectuant des calculs parallèles sur plusieurs processeurs, qui doivent éventuellement communiquer entre eux durant le traitement des tâches, il faut évidemment leur fournir assez d'énergie. Cependant ce problème dévient encore plus crucial dans les réseaux de capteurs où les nœuds ne disposent pas du tout de mémoire partagée, car ils sont obligés de s'envoyer des messages via la connexion sans fil qui peut vider leurs piles très vite. Dans quasiment toutes les application de réseaux de capteurs, les protocoles de communication utilisés sont spécialement élaborés pour minimiser le nombre de messages à envoyer, généralement les nœuds ne communiquent que dans les cas nécessaires.

4.3.3/ CONNEXION SANS FIL

Dans un réseau de capteurs, les nœuds utilisent des technologies de communication sans fil pour maintenir une connexion en consommant la moins d'énergie possible. Par exemple Zigbee [3] qui est basé sur le standard IEEE 802.15.4 [38].

Caractéristique technique	Bluetooth	Wifi	Zigbee
IEEE Spécification	802.15.1	802.11 a/b/g	802.15.4
Besoin de mémoire	250 Ko	1 Mo	4-32 Ko
Débit maximal	1 Mo/s	54 Mo/s	250 Ko/s
Portée théorique	10 m	100 m	10-100 m
Nombre de nœuds max.	7	32	65000+

TABLE 4.2 – Comparaison techniques des connexions sans fil

En faisant une comparaison technique de quelques protocoles sans fil [53, 92] (voir tableau 4.2), nous pouvons remarquer que Zigbee est un choix adéquat pour les réseaux de capteurs, car il peut supporter un très grand nombre de nœuds en consommant moins de ressources. Néanmoins, Zigbee n'est pas conçu pour créer un réseau de haut débit. La portée ratio théorique peut aller jusqu'à 100 mètres, mais en réalité, elle est restreinte à une dizaine de mètre pour des raisons de consommation et de durée de vie du réseau.

Quant à l'application du calcul parallèle dans les réseaux de capteurs, il faut prendre conscience qu'une telle connexion sans fil de faible puissance est loin d'être fiable. Elle souffre souvent de l'interférence qui peut causer des pertes et des corruptions de données. En outre, généralement la portée radio d'un nœud n'est pas assez longue pour couvrir entièrement le réseau, le calcul parallèle ne peut s'effectuer qu'entre des nœuds proches.

4.3.4/ TOLÉRANCE AUX PANNES

Comme pour beaucoup de systèmes informatiques, dans les réseaux de capteurs, la tolérance aux pannes doit aussi être prise en compte durant la conception des applications. En effet, elle permet au système d'effectuer des diagnostics automatiques et de se restaurer en cas de pannes sévères.

Un réseau de capteurs peut contenir un grand nombre de nœuds qui sont programmés pour travailler ensemble et atteindre un objectif commun. Les nœuds sont déployés aléatoirement et fonctionnent d'une manière autonome, lorsqu'un nœud tombe en panne, nous ne sommes pas censés nous rendre physiquement sur place pour le localiser et réparer. Il faut que les autres nœuds puissent détecter automatiquement des anomalies et exclure le nœud défectueux du réseau.

Un solution efficace et économique pour appliquer des mécanismes de tolérance aux pannes dans les réseaux de capteur est la redondance. Les nœuds déployés sont souvent très nombreux pour couvrir une zone la plus vaste possible. En outre, pour minimiser la consommation d'énergie et prolonger la durée de vie du réseau, les nœuds fonctionnent principalement d'une manière passive, c'est-à-dire qu'ils restent en mode inactive tant qu'ils ne détectent pas d'événement important. C'est la raison pour laquelle dans un tel réseau, il existe plein de nœuds libres pendant la plupart de leur durée de vie.

Il est donc possible d'utiliser plusieurs nœuds pour assurer une seule tâche. Lorsqu'ils détectent que le nœud détenant la tâche ne peut pas terminer son travail correctement, ils vont en choisir automatiquement un autre parmi eux qui va reprendre immédiatement le travail du nœud défectueux et continuer à travailler dessus (voir figure 4.6).

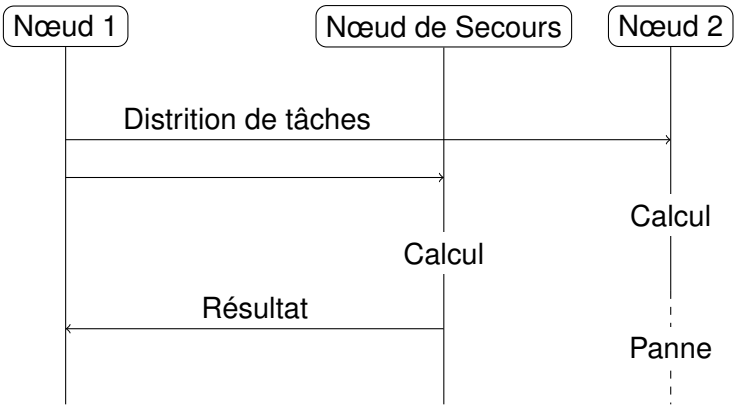


FIGURE 4.6 – Tolérance aux pannes avec la redondance de nœuds

La tolérance aux pannes est particulièrement indispensable pour le calcul parallèle et la sécurité, car ce dernier est une procédure qui demande une coopération cohérente et des échanges de données fiables entre les nœuds. Une moindre erreur durant le calcul et la transmission de données peut causer un échec complet du calcul. Par exemple, la pile du nœud est déchargée pendant le calcul, ou le nœud n'a pas assez d'énergie pour envoyer le résultat de calcul etc. Il faut prendre en considération toutes ces possibilité lors de la conception de l'application, et il faut que le système soit suffisamment robuste pour résister aux pannes éventuelles et réussir la mission dans un délai raisonnable.

4.4/ PARALLÉLISATION DES MULTIPLICATIONS SCALAIRES

Dans cette section, nous présentons notre méthode de calcul parallèle pour accélérer le calcul des multiplications scalaires dans les réseaux de capteurs.

Comme nous avons expliqué précédemment, les nœuds d'un réseau de capteurs ne possèdent aucune mémoire partagée. Tout échange de données doit passer par la communication radio qui est considérée comme étant la source principale de consommation d'énergie. D'ailleurs, les nœuds sont programmés avant le déploiement, et ils sont censés être déployés d'une manière aléatoire. Généralement les nœuds exécutent le même programme, et à cause de la limite de mémoire disponible, il est déconseillé d'installer des programmes volumineux qui prennent en considération toutes les circonstances possibles. Ainsi, pour paralléliser les multiplications scalaires, il faut mettre en place un algorithme unique qui demande très peu d'échange de données entre les nœuds participant au calcul parallèle, c'est-à-dire une parallélisation du type SISD (voir tableau 4.1).

Après la présentation des protocoles cryptographiques basés sur ECC, nous nous apercevons que la multiplication scalaire est l'opération la plus cruciale dont la complexité est un élément clé de la robustesse du cryptosystème. Elle est considérée comme une opération très coûteuse pour les systèmes embarqués malgré de nombreuse méthodes d'optimisation existantes. Sachant que les processeurs des capteurs sont souvent des

micro-contrôleurs de 8 ou 16 bits, alors que les nombres entiers à manipuler durant des calculs cryptographiques peuvent atteindre facilement quelques centaines de bits, il est donc indispensable d'implémenter en supplément des algorithmes d'arithmétique multi-précision, qui engendre une consommation de ressources très importante.

4.4.1/ DÉCOMPOSITION DES DONNÉES

L'objectif du calcul parallèle est de décomposer une tâche en plusieurs parties qui peuvent être traitées indépendamment et simultanément. Nous ne pouvons pas paralléliser des opérations comme dans [4], car il n'y a pas de mémoire partagée dans un réseau de capteurs. Le choix pertinent est donc la parallélisation des données, c'est-à-dire que les données sont décomposées en plusieurs portions qui sont distribuées aux nœuds exécutant un programme unique.

Notre méthode est conçue pour des applications événementielles dans lesquelles les nœuds demeurent inactifs pendant la plupart du temps pour minimiser la consommation d'énergie. Néanmoins, lorsqu'un événement crucial est survenu et détecté, il faut que le réseau le signale immédiatement à la station de base d'une manière sécurisée et sûre, même au détriment de la consommation d'énergie.

L'idée de la parallélisation est principalement basée sur [60], qui offre une décomposition de données efficace sans nécessiter de mémoire de partagée. L'objectif est de réduire le temps de calcul des multiplications scalaires en impliquant les nœuds de proximité dans le calcul. Nous supposons que la courbe elliptique est chargée sur les nœuds avant le déploiement, et le point de générateur G ne change pas durant toute la durée de vie du réseau.

Pour paralléliser une multiplication scalaire $Q = kG$ où G est le point générateur d'une courbe définie dans un corps premier fini, dénotée $E(\mathbb{F}_p)$, et k est un nombre entier de longueur l, qui peut être représenté en binaire de la manière ci-dessous :

$$k = \sum_{i=0}^{l-1} k_i 2^i \tag{4.1}$$

Le nœud qui décompose et distribue des données est le *nœud maître*, et les autres nœuds participant au calcul parallèle sont des *nœuds esclaves*.

Dans un premier temps, le maître décompose le nombre entier k en n segments S_i de longueur $b = \lceil l/n \rceil$ où n est le nombre de nœuds esclaves, et S_i peut être représenté en binaire comme dans la formule 4.2.

$$S_i = \sum_{j=ib}^{ib+b-1} k_j 2^j \tag{4.2}$$

Sachant que le point générateur G est choisi à priori, et il ne change pas pendant la durée de vie du réseau, le calcul des points $G_i = 2^{ib}G$ est donc possible. Le calcul de kG peut être décomposé comme suit :

$$\begin{cases} Q_0 & = S_0 G \\ Q_1 & = S_1 2^b G \\ \dots & \dots \\ Q_{n-1} & S_{n-1} 2^{b(n-1)} G \end{cases} \tag{4.3}$$

Puis le résultat final $Q = Q_0 + Q_1 + \dots Q_{n-1}$, et chaque Q_i peut être calculé indépendamment par un esclave.

Par exemple nous avons $k = k_{l-1}k_{l-2}\dots k_2 k_1 k_0$ où $k_i \in \{1, 0\}$, et il y a $n-1$ esclaves disponibles dans la portée radio du maître, le scalaire k est donc décomposé en n segments (voir la formule 4.4).

$$\begin{aligned} S_0 &= k_{b-1}k_{b-2}k_{b-3}\dots k_2 k_1 k_0 \\ S_1 &= k_{2b-1}k_{2b-2}k_{2b-3}\dots k_{b+2}k_{b+1}k_b \\ \dots &\quad \dots \\ S_{n-1} &= k_{nb-1}k_{nb-2}k_{nb-3}\dots k_{((n-1)b+2)}k_{((n-1)b+1)}k_{(n-1)b} \end{aligned} \tag{4.4}$$

Le maître en laisse une en sa mémoire locale, et les autres $n-1$ segments sont distribuées aux esclaves. Tous les nœuds préparent et stockent localement un ensemble de points

$$\begin{aligned} G_0 &= G \\ G_1 &= 2^b G \\ \dots &\quad \dots \\ G_{n-1} &= 2^{(n-1)} G \end{aligned} \tag{4.5}$$

Après la réception des données, les esclaves calculent chacun un Q_i en utilisant la formule 4.3. Une fois terminé, ils renvoient des Q_i au maître, qui va les combiner avec son propre résultat local pour obtenir le résultat final $Q = S_0 G + S_1 2^b G + S_2 2^{2b} G \dots S_{n-1} 2^{(n-1)} G$.

Cette méthode de décomposition de scalaire nous permet de découper le scalaire en n parties qui peut être traitée indépendamment, aucune échange de données n'est nécessaire entre esclaves. En outre, selon les paramètres recommandés [87], la taille du corps $\mathbb{F}_p$ dans lequel la courbe est définie est souvent comprise entre 112 et 521 bits, et il est possible de diffuser l'ensemble de S_i aux esclaves en une seule fois. Cependant, pour qu'un nœud puisse effectuer le calcul, il faut qu'il stocke en mémoire locale tous les points G_i précalculés, car il ne peut pas prévoir l'indice i du S_i qui lui sera affecté.

4.4.2/ PARALLÉLISATION SANS POINTS PRÉCALCULÉS

La méthode que nous avons présenté dans la section précédente nous permet de paralléliser le calcul des multiplications scalaires sur les courbes elliptiques, mais les nœuds participant doivent stocker en mémoire locale un ensemble de points précalculés pour pouvoir traiter indépendamment les tâches. Dans cette section, nous présentons une autre méthode qui peut paralléliser le calcul sans points précalculés.

Pour pouvoir paralléliser le calcul de $Q = kG$, le scalaire k est découpé en n segments S_i, et chacun doit être multiplié avec un point correspondant, noté $G_i = 2^{ib}G$ où i est l'indice du segment et b est sa longueur.

Néanmoins, cette méthode engendre une consommation de mémoire non négligeable et souffre de manque de flexibilité. Par exemple, les nœuds utilisent une courbe elliptique qui est définie dans un corps de taille 160, s'ils veulent paralléliser le calcul entre 4 nœuds, il faudra que tous les nœuds aient les points $2^{40}G$, $2^{80}G$ et $2^{120}G$ préparés et stockés en mémoire. Cependant s'il n'y a que 3 nœuds disponibles, il faudra donc avoir 2 nœuds qui traitent chacun une tâche de longueur 53, et le 3e nœud traite une tâche de longueur 54, et les points à préparer sont respectivement $2^{53}G$ et $2^{106}G$. Nous pouvons constater qu'il est difficile de définir et préparer un ensemble de points satisfaisant toutes les circonstances.

Nous proposons donc une méthode alternative qui peut préparer rapidement en temps réel les points nécessaire[30, 93]. La préparation d'un point 2^xG est aussi une multiplication scalaire qui peut être calculée en utilisant l'algorithme Doublement-et-Addition (voir algorithme 1). D'ailleurs, nous pouvons remarquer que

- L'algorithme Doublement-et-Addition ne fait que répéter le doublement de point tant qu'il n'y a pas de bit non-zéro ;
- Le même algorithme ne peut lire qu'un seul bit à chaque itération ;
- Le scalaire 2^x du point à préparer ne détient qu'*un seul bit non-zéro* qui est aussi le bit le plus significatif dans sa représentation binaire ;
- Le scalaire 2^x est plus court que la taille du corps dans lequel la courbe utilisée est définie.

Afin d'accélérer la préparation du point 2^xG, nous proposons d'utiliser l'algorithme de quadruplement (voir algorithme 10 où a est le paramètre de la courbe utilisée) qui nous permet de parcourir 2 bits à chaque itération, et l'algorithme 11 qui peut calculer directement le point 2^xG en donnant le paramètre x.

Algorithme 10 : Algorithme de quadruplement en coordonnées Jacobiennes

Données : $G'(X_1, Y_1, Z_1)$
Résultat : $2^2G'(X_4, Y_4, Z_4)$

1 $\alpha = 3X_1^2 + aZ_1^4$;
2 $\beta = \alpha^2 - 8X_1Y_1^2$;
3 $\gamma = -8Y_1^4 + \alpha(12X_1Y_1^2 - \alpha^2)$;
4 $\omega = 16aY_1^4Z_1^4 + 3\beta^2$;
5 $X_4 = -8\beta\gamma^2 + \omega^2$;
6 $Y_4 = -8\gamma^4 + \omega(12\beta\gamma^2 - \omega^2)$;
7 $Z_4 = 4Y_1Z_1\gamma$;

Les 2 algorithmes sont conçus pour le système de coordonnées Jacobiennes, avant de les utiliser, il faut d'abord convertir le point affine $G(X, Y)$ en point Jacobien $G'(X, Y, 1)$. Ces algorithmes nous permettent de préparer en temps réel les points nécessaires en fonction du nombre de tâches, et les nœuds n'ont plus besoin de stocker en mémoire l'ensemble de points précalculés.

Nous supposons que la courbe utilisée est définie dans un corps de 160 bits, pour pouvoir paralléliser le calcul entre 2, 3 et 4 nœuds, il faut stocker dans chaque nœud 6 points précalculés : $2^{120}G, 2^{80}G, 2^{54}G, 2^{53}G, 2^{40}G$ et G. Si chaque point est représenté par 2 coordonnées affines, il faudra donc $(160 \times 2 \times 6)/8 = 240$ octets pour les stocker.

Sachant que généralement les systèmes embarqués ne détiennent qu'une mémoire de

Algorithme 11 : Algorithme de $2^xG'$ en coordonnées Jacobiennes

Données : $G'(X_1, Y_1, Z_1)$, $x \in \mathbb{Z}^+$
Résultat : $2^xG'(X_{2^x}, Y_{2^x}, Z_{2^x})$

$\alpha_1 = X_1$;
$\beta_1 = 3X_1^2 + a$;
$\gamma_1 = -Y_1$;
pour $i \in [2, x]$ **faire**
    $\alpha_i = \beta_{i-1}^2 - 8\alpha_{i-1}\gamma_{i-1}^2$;
    $\beta_i = 3\alpha_i^2 + 16^{i-1}a(\prod_{j=1}^{i-1}\gamma_j)^4$;
    $\gamma_i = -8\gamma_{i-1}^4 - \beta_{i-1}(\alpha_i - 4\alpha_{i-1}\gamma_{i-1}^2)$;
fin
$\omega_x = 12\alpha_x\gamma_x^2 - \beta_x^2$;
$X_{2^x} = \beta_x^2 - 8\alpha_x\gamma_x^2$;
$Y_{2^x} = 8\gamma_x^4 - \beta_x\omega_x$;
$Z_{2^x} = 2x\prod_{i=1}^{x}\gamma_i$;

quelques dizaines de kilo octets, la quantité de mémoire requise pour le stockage de points précalculés est donc non négligeable. Pour évaluer leur aptitude à l'utilisation, les résultats de test de performance sont donnés dans la section 4.6, car il faut que les algorithmes soient assez efficaces pour ne pas ralentir la procédure du calcul parallèle.

4.4.3/ PROTOCOLE

Nous avons déjà présenté les algorithmes qui nous permettent de décomposer les tâches de calcul, et dans cette section, nous présentons donc le protocole qui est chargé de la coordination de la communication entre les nœuds.

Nous supposons que de déploiement de capteurs est basé sur les clusters (voir figure 4.7), car une telle topologie est prouvée plus efficace en termes de consommation d'énergie [36, 35]. D'ailleurs, un algorithme de cryptographie symétrique peut être appliqué pour sécuriser la distribution et la récupération des données. Nous avons déjà expliqué dans la section 2.4.4.1 que la gestion de clés dans un cluster peut être un choix adéquat pour la cryptographie symétrique.

Chaque cluster dispose au maximum d'un cluster-head qui est élu par les autres membres du cluster. Les membres d'un cluster ne sont pas censés communiquer directement avec les membres des autres clusters, ils ne dialoguent qu'avec leur cluster-head, qui est chargé de la communication inter-cluster. Comme la plupart des communications passent par le cluster-head, le cluster doit changer périodiquement le cluster-head pour équilibrer la consommation d'énergie intérieure.

Nous supposons que dans une application de surveillance, les nœuds sont déployés et regroupés dans des clusters. Les membres sont programmés pour collecter et envoyer périodiquement des données au cluster-head, qui les agrège, compresse, chiffre et envoie à la station de base (voir figure 4.8).

Lorsque le cluster-head détecte des données révélant un évènement critique, qui doit être signalé immédiatement à la station de base d'une manière sécurisée, nous pouvons demander aux autres membres disponibles du même cluster de participer au calcul

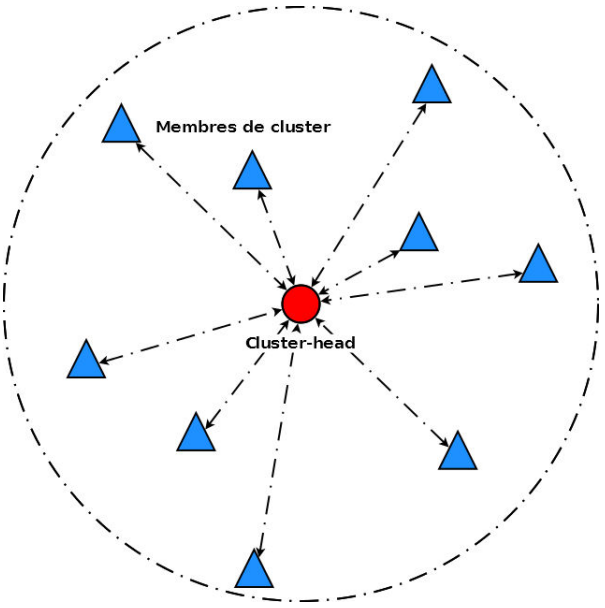


FIGURE 4.7 – Clusters dans un réseau de capteurs

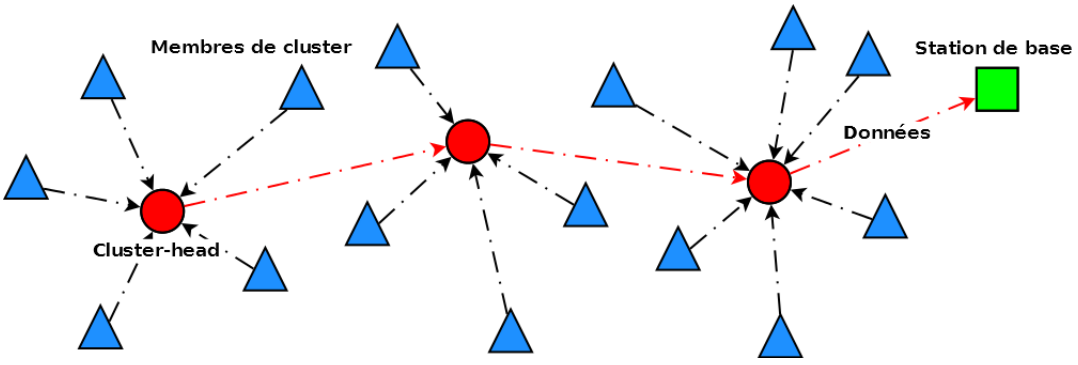


FIGURE 4.8 – Collection de données dans un réseau de capteurs

cryptographique pour accélérer la procédure de chiffrement, qui est représentée graphiquement dans la figure 4.9 dont les explications sont données ci-dessous :

1. Les membres du cluster envoient périodiquement des données collectées au cluster-head qui est chargé de traitement d'informations (agrégation, compression, chiffrement etc.) ;
2. Le cluster-head détecte un événement important, et il veut utiliser le parallélisme pour accélérer le chiffrement des informations sensibles. Pour demander de l'aide aux autres membres, il leur diffuse une demande de participation pour identifier les nœuds disponibles dans le cluster ;
3. Après la réception de la demande, un membre retourne une réponse $\{i, a\}$ où i est son identifiant et r est une valeur booléenne représentant sa disponibilité ;
4. Le cluster-head sélectionne, parmi les membres disponibles, les nœuds qui vont participer au calcul parallèle. Les nœuds sélectionnés deviennent les *nœuds esclaves*, alors que le cluster-head devient le *nœud maître*. Le maître décompose d'abord le calcul en n parties indépendantes, ensuite il diffuse aux eslaves les

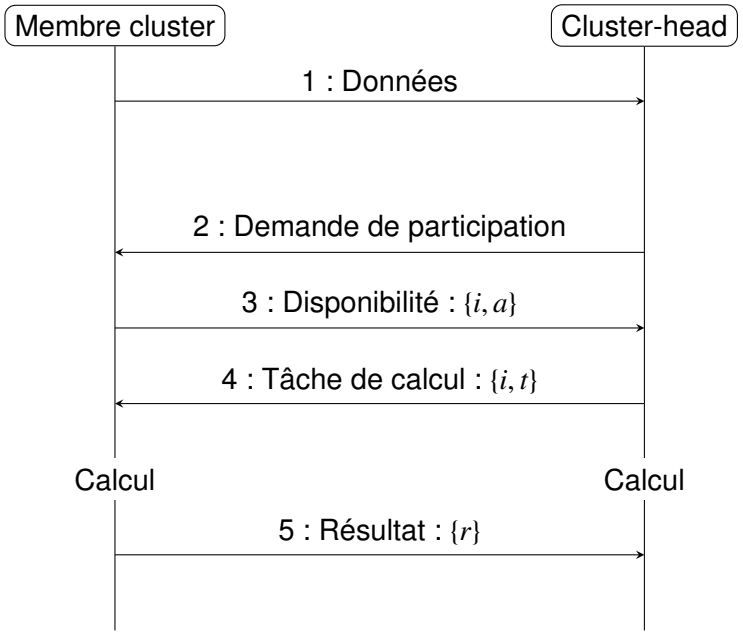


FIGURE 4.9 – Procédure de multiplication scalaire parallèle

tâches de calcul $\{\{i_1, t_1\}, \{i_2, t_2\}, \ldots, \{i_{n-1}, t_{n-1}\}\}$ où t_i représente les tâches confiées et n est le nombre de nœuds sélectionnés.

5. Les nœuds disposent maintenant les informations nécessaires pour effectuer le traitement des tâches, une fois terminé, les esclaves renvoient les résultats $\{r_1, r_2, \ldots, r_{n-1}\}$ à leur maître. Ce dernier combine l'ensemble de r_i pour obtenir le résultat final de la multiplication scalaire R, qui peut ensuite être utilisé pour chiffrer ou signer le message à envoyer.

Après l'étude du protocole, nous pouvons remarquer que les nœuds doivent s'envoyer un certain nombre de messages pour procéder au calcul parallèle. La gestion des communications sans fil dans les réseaux de capteurs est une procédure compliquée qui implique l'utilisation des autres protocoles, e.g. le protocole de gestion d'accès au médium qui a pour objectif d'éviter des collisions durant les transmissions des paquets. Cependant, pendant nos travaux de recherche, nous nous centralisons uniquement sur le protocole de la couche application.

4.5/ ARITHMÉTIQUE MULTIPRÉCISION

Selon les paramètres recommandés [87], les courbes elliptiques utilisées doivent être définies dans un corps fini dont la taille est comprise entre 112 et 521 bits, il faut donc que les nœuds puissent gérer l'arithmétique modulaire et des nombres entiers de très grande taille. Cependant, les systèmes embarqués sont souvent équipés des processeurs de 8 et 16-bit, qui ne peuvent pas gérer directement de grandes valeurs dont la longueur dépasse largement leur capacité. Des algorithmes particuliers sont donc indispensables pour implémenter les protocoles cryptographiques [34].

4.5.1/ ADDITION ET SOUSTRACTION

Nous supposons la longueur de mot de notre processeur est W. υ est un grand nombre entier quelconque, et $\upsilon \in [2^W, \infty[$. Sa longueur en nombre de bits est $m = \lceil \log_2 \upsilon \rceil$, et celle en nombre de mots est $t = \lceil m/W \rceil$. Ainsi, pour stocker υ en mémoire, il faut utiliser un tableau $A[i]$ où $i \in [0, t-1]$ et la longueur totale $m = Wt$ bits. L'affectation de valeur d'un élément peut s'écrire sous forme d'un couple $(\varepsilon, z) \leftarrow \omega$ dont

$$\begin{array}{lll} z & \leftarrow \omega \bmod 2^W & \\ \varepsilon & \leftarrow 0 \quad \text{si } \omega \in [0, 2^W[, & \text{sinon } \varepsilon \leftarrow 1 \end{array} \tag{4.6}$$

Les algorithmes 12 et 13 montrent les opérations à effectuer pour additionner et soustraire 2 grands nombres entiers A et B qui sont représentés de la manière ci-dessus. Nous supposons que $A, B \in [0, 2^{Wt} - 1]$ et leurs longueurs en nombre de mots t sont identique. Le bit de retenue $\varepsilon = 1$ lorsque le résultat $C \notin [0, 2^{Wt} - 1]$.

Algorithme 12 : Addition multiprécision $A + B \pmod{2^{Wt}}$

Données : $A, B \in [0, 2^{Wt}[$
Résultat : (ε, C) où $C = A + B \bmod 2^{Wt}$ et ε est le bit de retenue

```
1 (ε, C[0]) ← A[0] + B[0];
2 pour i ← 1 a (t − 1) faire
3 |   (ε, C[i]) ← A[i] + B[I] + ε;
4 fin
5 retourner (ε, C)
```

Algorithme 13 : Soustraction multiprécision $A - B \pmod{2^{Wt}}$

Données : $A, B \in [0, 2^{Wt}[$
Résultat : (ε, C) où $C = A - B \bmod 2^{Wt}$ et ε

```
1 (ε, C[0]) ← A[0] − B[0];
2 pour i ← 1 a (t − 1) faire
3 |   (ε, C[i]) ← A[i] − B[i] − ε
4 fin
5 retourner (ε, C)
```

Pendant nos travaux de recherche, nous avons choisi des courbes qui sont définies dans un corps premier fini, noté $\mathbb{F}_p$, et toute les valeurs à gérer sont donc comprises entre 0 et $p - 1$. Le calcul de l'addition et de la soustraction dans $\mathbb{F}_p$ est détaillé dans les algorithmes 15 et 16.

Nous pouvons constater qu'ils sont basés sur les 2 premiers algorithmes, à l'issue de l'exécution, nous devons tester si la valeur $C \in [0, p-1]$. Si $C < 0$, alors $C = C + p$, mais $C = C - p$ si $C \geq p$.

4.5.2/ MULTIPLICATION

La multiplication multiprécision est un peu plus compliquée (voir algorithme 17), mais le principe arithmétique reste le même, au lieu de parcourir les chiffres des 2 opérandes,

Algorithme 15 : Addition multiprécision $A + B \pmod{p}$

Données : p et $A, B \in [0, p-1]$
Résultat : $C = A + B \pmod{p}$
1 $(\varepsilon, C) \leftarrow$ Algorithme 12 où $C = A + B \pmod{2^{Wt}}$;
2 **si** $\varepsilon = 1$ **alors**
3 $\quad C \leftarrow C - p$;
4 **sinon si** $C \geq p$ **alors**
5 $\quad C \leftarrow C - p$;
6 **retourner** C

Algorithme 16 : Soustraction multiprécision $A - B \pmod{p}$

Données : p et $A, B \in [0, p-1]$
Résultat : $C = A - B \pmod{p}$
1 $(\varepsilon, C) \leftarrow$ Algorithme 13 où $C = A - B \pmod{2^{Wt}}$;
2 **si** $\varepsilon = 1$ **alors**
3 $\quad C \leftarrow C + p$;
4 **retourner** C

nous parcourons les éléments des 2 tableaux $A[i], B[i]$ où $i \in [0, t-1]$. La longueur du résultat $C[i]$ peut aller jusqu'à $2t$. (UV) est une nombre de longueur $2W$, car c'est le produit de 2 opérandes de longueur W (voir ligne 7 de l'algorithme 17), mais seulement les W bits de poids fort sont stockés dans U.

Algorithme 17 : Multiplication multiprécision $C = A \cdot B$ où $A, B \in [0, p-1]$

Données : $A, B \in [0, p-1]$
Résultat : $C = A \cdot B$
1 **pour tous les** $i \in [0, t-1]$ **faire**
2 $\quad C[i] \leftarrow 0$;
3 **fin**
4 **pour tous les** $i \in [0, t-1]$ **faire**
5 $\quad U \leftarrow 0$;
6 $\quad$ **pour tous les** $j \in [0, t-1]$ **faire**
7 $\quad\quad (UV) \leftarrow C[i+j] + A[i] \cdot B[j] + U$;
8 $\quad\quad C[i+j] \leftarrow V$;
9 $\quad$ **fin**
10 $\quad C[i+t] \leftarrow U$;
11 **fin**
12 **retourner** C

Après la multiplication, le résultat C doit encore passer à la réduction modulo p dont l'algorithme utilisé est présenté dans plus bas dans la section suivante. L'algorithme de division n'est pas donné, car dans les cryptosystèmes qui sont basés sur les courbes elliptiques, nous choisissons les systèmes de coordonnées permettant d'éviter l'inverse modulaire (voir algorithme 2) qui est considéré comme une opération extrêmement coûteuse.

4.5.3/ RÉDUCTION MODULO

Comme les courbes elliptiques utilisées sont définies dans un corps premier fini $\mathbb{F}_p$, nous devons donc effectuer des opérations $r \equiv z \pmod{p}$ tout au long du calcul des multiplications scalaires. Nous avons choisi l'algorithme de réduction Barrett [5] qui permet d'estimer le quotient de $\lfloor z/p \rfloor$ avec des opérations moins coûteuses.

L'idée de la réduction barrett est très simple. Nous supposons que tous les nombres sont représenté en base b, $k = \lfloor \log_b p \rfloor + 1$ est le plus petit nombre tel que $b^k > p$, puis $z \bmod p$ peut être calculé en utilisant l'algorithme 18.

Algorithme 18 : Réduction Barrett

Données : p, $b \geq 3$, $z \in [0, b^{2k} - 1]$, $\mu = \lfloor b^{2k}/p \rfloor$
Résultat : $z \bmod p$

$\hat{q} \leftarrow \lfloor \lfloor z/b^{k-1} \rfloor \cdot \mu / b^{b+1} \rfloor$;
$r \leftarrow (z \bmod b^{k+1}) - (\hat{q} \cdot p \bmod b^{b+1})$;
si $r < 0$ **alors**
  $r \leftarrow r + b^{b+1}$;
fin
tant que $r \geq p$ **faire**
  $r \leftarrow r - p$;
fin
retourner r

Quelques caractéristiques de l'algorithme qui nous permettent d'accélérer le calcul de modulo p sont listées ci-dessous :

- Généralement la taille p du corps dans lequel la courbe est définie et la base b ne changent pas pendant toute la durée de vie du système, donc nous pouvons stocker et réutiliser la valeur de $\mu = \lfloor b^{2k}/p \rfloor$;
- La base choisie $b = 2^L$ où L est souvent la longueur de mot du processeur, par exemple, si nous essayons d'implémenter l'algorithme sur un processeur de 16-bit, alors $L = 16$;
- Les calculs de $\lfloor z/b^{k-1} \rfloor$ et de $\lfloor \mu/b^{k+1} \rfloor$ peuvent être réalisés en faisant simplement un décalage de bits vers la droite. Par exemple, pour calculer $1011010111_2/1000_2$, il suffit que 1011010111_2 décale 3 bits vers la droite, et le résultat de la division est 1011010_2.

Une autre méthode qui nous permet de calculer encore plus efficacement le modulo p est proposée dans [10]. L'idée est basée sur les paramètres recommandés de NIST [1] qui propose un ensemble de paramètres pour configurer proprement les courbes elliptiques utilisées dans de cryptosystèmes.

$$\begin{aligned} p_{192} &= 2^{192} - 2^{64} - 1 \\ p_{224} &= 2^{224} - 2^{96} + 1 \\ p_{256} &= 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1 \\ p_{384} &= 2^{384} - 2^{128} - 2^{96} + 2^{32} - 1 \\ p_{521} &= 2^{521} - 1 \end{aligned} \tag{4.7}$$

5 corps premier fini sont donnés (voir formule 4.7), et à part celui de p_{521}, les exposants

1. National Institute of Standards and Technology

dans les autre polynômes sont des multiples de 16, et cette propriété rend l'algorithme de réduction modulo extrêmement efficace sur des processeurs de 16-bit.

Par exemple, nous voulons calculer $c \bmod p$ où $p = p_{192} = 2^{192} - 2^{64} - 1$ et $c \in [0, p^2 - 1]$. Le nombre c peut être écrit sous forme d'un polynôme en base 32, comme la formule 4.8.

$$\begin{aligned} c = \quad & c_{11}2^{352} + c_{10}2^{320} + c_9 2^{288} + c_8 2^{256} + c_7 2^{224} + c_6 2^{192} + \\ & c_5 2^{160} + c_4 2^{128} + c_3 2^{96} + c_2 2^{64} + c_1 2^{32} + c_0 \end{aligned} \tag{4.8}$$

Nous pouvons calculer le modulo p des puissances de 2 dont les exposants sont supérieurs à 192 avec la congruence (voir formule 4.9).

$$\begin{aligned} 2^{192} &\equiv 2^{64} + 1 \pmod{p} \\ 2^{224} &\equiv 2^{96} + 2^{32} \pmod{p} \\ 2^{256} &\equiv 2^{128} + 2^{64} \pmod{p} \\ 2^{288} &\equiv 2^{160} + 2^{96} \pmod{p} \\ 2^{320} &\equiv 2^{128} + 2^{64} + 1 \pmod{p} \\ 2^{352} &\equiv 2^{160} + 2^{96} + 2^{32} \pmod{p} \end{aligned} \tag{4.9}$$

Ainsi, le calcul de $c \bmod p_{192}$ peut être réécrit comme la formule 4.10, et les coefficients du polynôme sont aussi donnés dans le tableau 4.3.

2^{160}	2^{128}	2^{96}	2^{64}	2^{32}	2^0
c_{11}		c_{11}		c_{11}	
	c_{10}		c_{10}		c_{10}
c_9		c_9			
	c_8		c_8		
		c_7		c_7	
			c_6		c_6
c_5	c_4	c_3	c_2	c_1	c_0

TABLE 4.3 – Répartition des coefficients du polynôme $c \bmod p_{192}$

$$\begin{aligned} c \equiv \quad & c_{11}2^{160} + c_{11}2^{96} + c_{11}2^{32} \\ & + c_{10}2^{128} + c_{10}2^{64} + c_{10} \\ & + c_9 2^{160} + c_9 2^{96} \\ & + c_8 2^{128} + c_8 2^{64} \\ & + c_7 2^{96} + c_7 2^{32} \\ & + c_6 2^{64} + c_6 \\ & + c_5 2^{160} + c_4 2^{128} + c_3 2^{96} + c_2 2^{64} + c_1 2^{32} + c_0 \pmod{p} \end{aligned} \tag{4.10}$$

La procédure de calcul est présentée dans l'algorithme 19. Nous pouvons constater que pour calculer $c \bmod p_{192}$ où $c \in [0, p_{192}^2 - 1]$, nous découpons d'abord c en 12 segments de 32 bits $c_i, i \in [0, 11]$, qui peuvent reconstituer ensuite 8 nombres de 192 bits $s_j, j \in [0, 7]$. Enfin, pour terminer, il suffit d'additionner tous ces nombre $\sum_{j=1}^{7} s_j$. Après chaque addition, nous comparons la somme temporaire s_{tmp} avec p_{192}, si $s_{tmp} > p_{192}$, nous devrons faire $s_{tmp} - p_{192}$ avant de passer à l'addition suivante.

Nous nous apercevons que cette méthode ne nécessite aucun calcul complexe, il suffit d'effectuer quelques opérations bit-à-bit et un certain nombre de soustractions pour

Algorithme 19 : Réduction rapide de modulo $p_{192} = 2^{192} - 2^{64} - 1$

Données : $c = (c_{11}, c_{10}, \ldots, c_1, c_0)$ en base 32, et $c \in [0, p^2 - 1]$
Résultat : $c \bmod p_{192}$

1 Définir un ensemble de nombres entiers de longueur 192 : ;
2 $s_1 = (c_{11}, 0, c_{11}, 0, c_{11}, 0)$;
3 $s_2 = (0, c_{10}, 0, c_{10}, 0, c_{10})$;
4 $s_3 = (c_9, 0, c_9, 0, 0, 0)$;
5 $s_4 = (0, c_8, 0, c_8, 0, 0)$;
6 $s_5 = (0, 0, c_7, 0, c_7, 0)$;
7 $s_6 = (0, 0, 0, c_6, 0, c_6)$;
8 $s_7 = (c_5, c_4, c_3, c_2, c_1, c_0)$;
9 **retourner** $(\sum_{i=1}^{7} s_i \bmod p_{192})$

calculer le modulo dans un corps premier de 192-bit. Cependant cette méthode ne fonctionne qu'avec les corps proposés par NIST, et ne permet pas de calculer le modulo dans un corps premier fini quelconque.

4.6/ EVALUATION DE PERFORMANCE

Afin d'évaluer la performance de notre méthode de parallélisme, nous l'avons implémentée sur des capteurs Telosb (voir figure 4.10), un modèle conçu par Crossbow Technology pour des usages de recherche, et les caractéristiques techniques principales de la plate-forme sont données dans le tableau 4.4.

FIGURE 4.10 – Plate-forme Telosb de Crossbow Technology

Nous pouvons constater qu'elles correspondent bien aux nœuds dont nous parlons dans la littérature : une puissance de calcul relativement faible, accompagnée d'une mémoire très limitée.

Processeur :	8 MHz MSP 430 16-bit MCU
Antenne Radio :	CC2420 (802.15.4/Zigbee)
ROM :	48 Ko
RAM :	10 Ko

TABLE 4.4 – Caractéristiques techniques de plate-forme Telosb

Nous avons choisi nesC [29] comme langage de programmation, qui signifie **n**etwork **e**mbedded **s**ystem **C** en Anglais, c'est un langage spécifiquement créé pour la pro-

grammation événementielle, c'est aussi le langage de développement par défaut sous Tinyos [57]. Un des avantages d'un tel programme est que son exécution est entièrement pilotée par des événements, tant qu'il n'y a pas d'événement important, l'exécution est automatiquement mis en pause, et cette propriété permet au nœud de minimiser la consommation d'énergie, qui est considérée comme un des facteurs cruciaux pendant la conception d'une application pour les réseaux de capteurs.

Tinyos est un système d'exploitation développé et maintenu par l'Université de Berkeley, l'objectif du projet est d'insérer une couche de virtuelle pour faciliter la communication entre la couche matérielle et la couche application. Certes, nous pouvons créer des applications en assembleur et en C qui manipulent directement des composants matériels. Cependant, premièrement, elles sont souvent difficile à maintenir, car il faut que le développeur dispose non seulement des compétences en informatique, mais aussi des connaissances fondamentales en électronique. Deuxièmement, ce genre de programmes est moins portable. L'avantage principale de Tinyos est que les programmes ne sont pas censés accéder directement aux composants matériels, toute communication doit passer par des interfaces fournies par Tinyos, qui est chargé de l'association entre le programme et le pilote correspondant en fonction de la plate-forme utilisée. Prenons l'exemple du langage Java, les programmes sont compilés pour générer des bytecodes, qui peuvent être ensuite interprétés par des machines virtuelles différentes.

Comme le but de nos travaux de recherche est d'accélérer le calcul des multiplications scalaires sur les courbes elliptiques, notre programme essaie donc de paralléliser la tâche de calcul $Q = kG$ entre plusieurs nœuds voisins, le temps de calcul et la consommation d'énergie sont respectivement mesurés et estimée. Afin d'avoir une réduction modulo efficace, les nœuds utilisent la même courbe qui est définie dans le corps premier fini $NIST_{192}$ [10] dont les paramètres recommandés sont donnés dans le tableau 4.5. a et b sont les paramètres de la forme de Weierstrass simplifiée (voir formule 3.3), (x_G, y_G) est le point de générateur dont l'ordre est n. Le scalaire k utilisé est un nombre entier de longueur 160, qui est censé pouvoir fournir le même niveau de sécurité qu'une clé de 1024 bits avec RSA.

Paramètre	Valeur recommandée
p	$2^{192} - 2^{64} - 1$
a	-3
b	`0x 6421 0519 e59c 80e7 0fa7 e9ab 7224 3049 feb8 deec c146 b9b1`
x_G	`0x 188d a80e b030 90f6 7cbf 20eb 43a1 8800 f4ff 0afd 82ff 1012`
y_G	`0x 0719 2b95 ffc8 da78 6310 11ed 6b24 cdd5 73f9 77a1 1e79 4811`
n	`0x ffff ffff ffff ffff ffff ffff 99de f836 146b c9b1 b4d2 2831`

TABLE 4.5 – Paramètres recommandés du standard $NIST_{192}$

Les nœuds sont déployés dans une zone de $10m \times 10m$ pour former un cluster dont le cluster-head est connecté directement à un PC (voir figure 4.11), une fois allumé, les nœuds continuent à répéter des multiplications scalaires parallèles avec des valeurs de k aléatoires. La communication à l'intérieur du cluster, comme la distribution de tâches et la récupération des résultats sont sécurisées avec Trivium [21], un algorithme de cryptographie symétrique très léger qui est capable de chiffrer et déchiffrer des messages en moins de 1 ms.

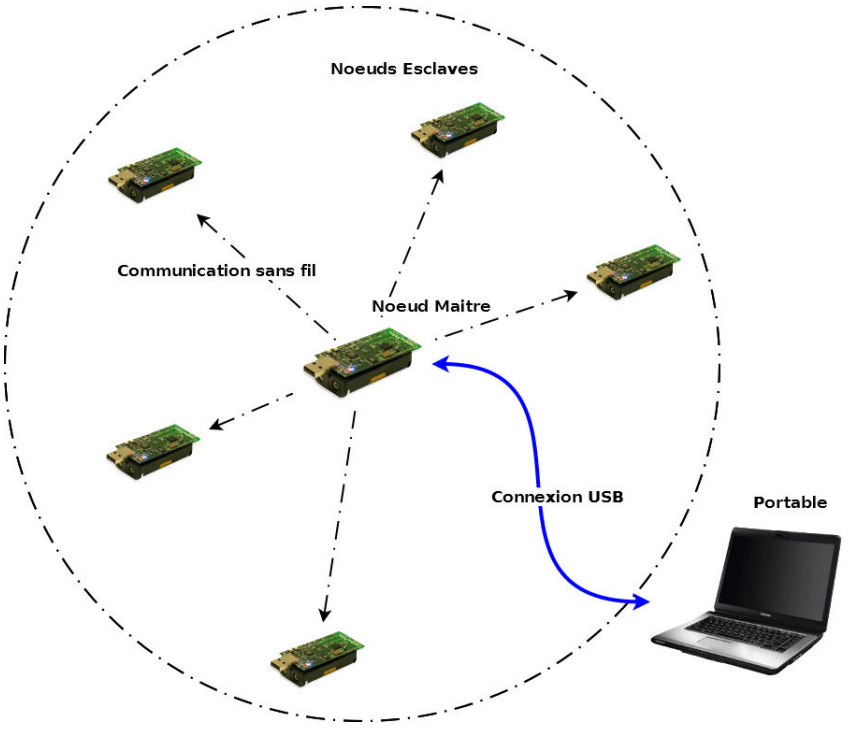


FIGURE 4.11 – Déploiement des nœuds pour le test de performance

4.6.1/ PARALLÉLISATION AVEC POINTS PRÉCALCULÉS

La première partie du test consiste à évaluer la performance de notre méthode de parallélisme avec des points précalculés et stockés en mémoire locale. Les temps de calcul en millisecondes avec et sans calcul parallèle sont donnés dans le tableau 4.6. Il est difficile de comparer les valeurs absolues des résultats avec celles des autres implémentations à cause des différences de technique utilisé et de scénario de test. Nous nous intéressons seulement aux gains obtenus.

Nombre de nœuds	Affine	Gain	Jacobien	Gain
1	2307.27		1003.55	
2	1189.96	48.43%	549.71	45.22%
3	861.48	62.66%	424.60	57.69%
4	665.68	71.15%	342.51	65.87%
5	583.29	74.72%	309.93	69.12%
6	581.32	74.80%	311.87	68.92%

TABLE 4.6 – Temps de calcul (ms) de notre méthode de parallélisme

Nous pouvons voir dans la figure 4.12 que le temps de calcul diminue progressivement lorsque plus de nœuds participent au calcul parallèle. Nous supposons que le temps de calcul avec p nœuds est T_p, nous pouvons encore évaluer la performance de notre méthode avec son speedup S_p qui est défini dans la formule 4.11, et les résultats de speedup sont donnés dans le tableau 4.7 et représentés graphiquement dans la figure 4.13.

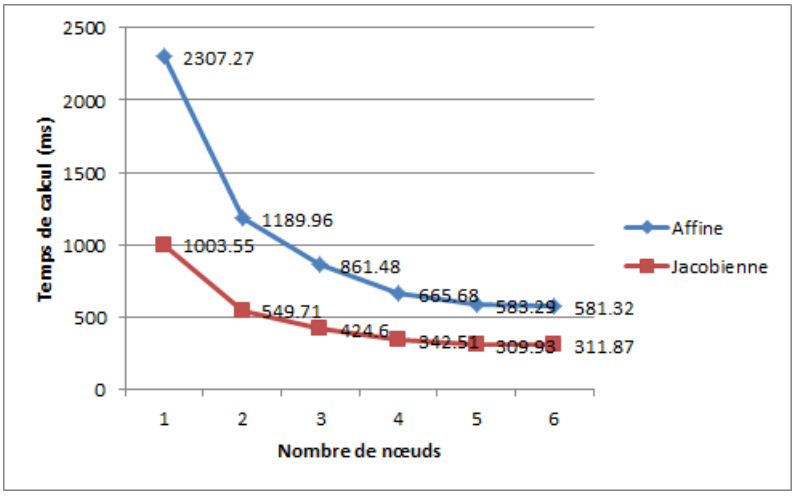


FIGURE 4.12 – Temps de calcul (ms) de notre méthode de parallélisme

$$S_p = \frac{T_1}{T_p} \tag{4.11}$$

Nombre de nœuds :	1	2	3	4	5	6
Affine :	1.00	1.94	2.68	3.47	3.96	3.97
Jacobien :	1.00	1.83	2.36	2.93	3.24	3.22

TABLE 4.7 – Speedup de notre méthode de parallélisme

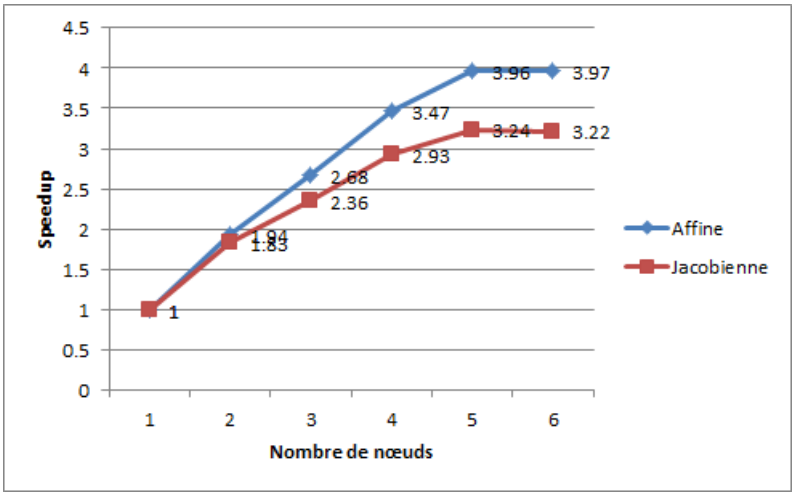


FIGURE 4.13 – Speedup $S_p = \frac{T_1}{T_p}$ de notre méthode de parallélisme

Nous pouvons remarquer une baisse d'accélération importante dès que nous utilisons plus de 5 nœuds. Car le cluster-head a besoin de temps supplémentaire, nommé surcoût, pour coordonner les communications radio, combiner les résultats reçus et calculer le résultat final. Les surcoûts mesurés durant notre expérimentation sont présentés dans le tableau 4.8 et la figure 4.14. Nous nous apercevons qu'à partir du moment où nous utilisons plus de 5 nœuds, il y a une augmentation significative de surcoût. Ainsi, *nous*

pouvons donc en conclure que le nombre de nœuds participant au calcul parallèle doit être limité à moins de 5.

Nombre de nœuds :	1	2	3	4	5	6
Surcoût moyen :	0.00	36.33	92.39	88.86	121.84	196.78

TABLE 4.8 – Surcoût (ms) moyen de notre méthode de parallélisme

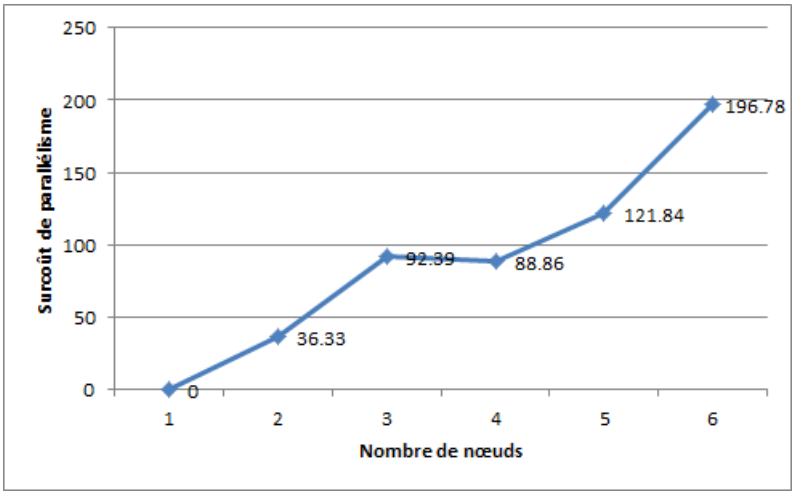


FIGURE 4.14 – Surcoût (ms) de notre méthode de parallélisme

Pendant notre test, la courbe elliptique utilisée est définie dans un corps de 192-bit, et les points précalculés et stockés sont représentés en coordonnées Affines (x, y) qui nous permettent de minimiser la consommation de mémoire, car chaque point est représenté par 2 nombres entiers de 192-bit, et il peut être transformé facilement à un point en coordonnées Jacobiennes $(x, y) \rightarrow (x, y, 1)$ qui offre généralement un calcul plus efficace.

Nombre de nœuds :	1	2	3	4	5	6
Mémoire :	0	48	96	144	192	240

TABLE 4.9 – Mémoire nécessaire (octet) pour stocker des points précalculés

La mémoire nécessaire pour stocker des points précalculés sont donnée dans le tableau 4.9. Les plate-formes Telosb [18] et Micaz [19] disposent respectivement d'une mémoire de données de 48 Ko et 128 Ko, qui est largement suffisante pour stoker les points.

Un des inconvénients principaux du parallélisme est la consommation d'énergie, et ce problème devient plus crucial pour les réseaux de capteurs, car les piles des nœuds ne peuvent pas être changées après le déploiement, et nous accélérons le traitement de tâches en détriment de la durée de vie du système. Nous avons déjà présenté précédemment que durant le calcul parallèle, plus de nœuds sont impliqués dans le calcul, et par conséquent, plus d'énergie est consommée car les nœuds doivent échanger des données entre eux.

Il est difficile de mesurer précisément la consommation d'énergie de chaque nœud, généralement dans la littérature, nous utilisons des simulateurs pour estimer la consommation d'énergie. Avrora [102] est un simulateur qui est largement utilisé dans la communauté de

recherche. Il est développé pour simuler le comportement des plate-formes de la famille Mica, e.g. Mica1, Mica2 et MicaZ. Nous l'avons utilisé pour évaluer la consommation d'énergie de notre méthode, c'est vrai qu'il n'est pas initialement conçu pour la plate-forme Telosb, mais les résultats produits (voir tableau 4.10) sont déjà suffisants pour étudier l'impact de notre méthode sur le réseau.

Nombre de nœuds :	1	2	3	4	5	6
Énergie :	0.889	2.125	3.156	4.192	5.225	6.256

TABLE 4.10 – Consommation d'énergie (Joule) de notre méthode de parallélisme

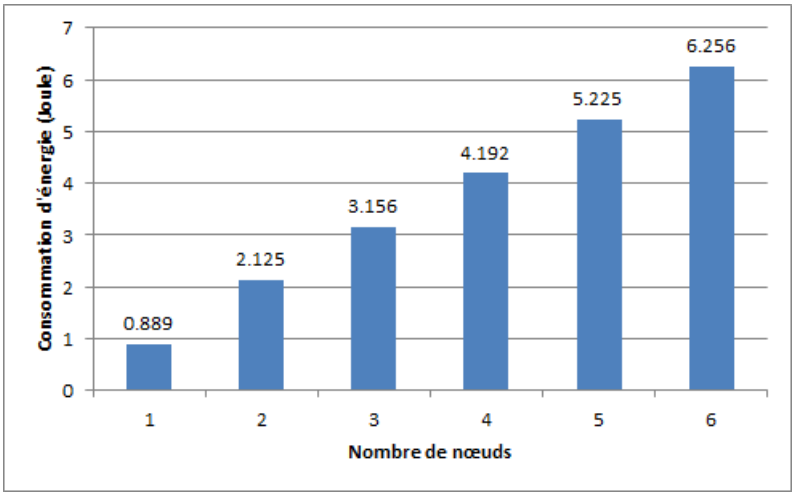


FIGURE 4.15 – Consommation d'énergie (Joule) de notre méthode de parallélisme

Dans la figure 4.15, nous pouvons constater que lorsque le calcul est effectué sur un seul nœud, très peu d'énergie est consommée. Cependant, quand le calcul parallèle est utilisé, comme les esclaves doivent recevoir des tâches de leur maître et lui renvoyer des résultats, il y a une augmentation importante de la consommation d'énergie.

Pour conclure la première partie de notre test, nous avons parallélisé le calcul des multiplications scalaires en utilisant jusqu'à 6 nœuds, et nous avons obtenu un gain maximal de 74.8%. Néanmoins, les résultats de simulation ont montré que le calcul parallèle consomme trop d'énergie par rapport au calcul local. Ainsi, une utilisation abusive d'une telle technique risque de réduire la durée de vie du réseau, et nous proposons donc de n'utiliser le calcul parallèle que dans des cas urgents, par exemple, la détection d'un événement crucial.

4.6.2/ PARALLÉLISATION SANS POINTS PRÉCALCULÉS

La deuxième partie consiste à tester la performance de la méthode qui nous permet de préparer en temps réel les points 2^xG dont nous avons besoin le calcul parallèle et de ne pas les stocker en permanence. L'idée est d'utiliser des algorithmes qui peuvent *théoriquement* calculer plus rapidement que Doublement-et-Addition les points pour le traitement de tâche. La configuration de la courbe elliptique est exactement la même que la première partie (voir tableau 4.5).

Nombre de nœuds :	2	3	4	5
Point calculé :	$2^{80}G$	$2^{53}G$	$2^{40}G$	$2^{32}G$
Algorithme de 2^xG :	1293	654	421	301
Algorithme de 2^2G :	465	297	228	182

TABLE 4.11 – Temps de calcul (ms) pour préparer des points 2^xG

Les temps de calcul utilisant l'algorithme de 2^xG (11) et l'algorithme de 2^2G (10) sont donnés dans le tableau 4.11. Par exemple, nous voulons paralléliser le calcul de kG entre 2 nœuds où k est un nombre entier de 160 bits qui est décomposé en 2 segments s_1 et s_2, et $k = 2^{80}s_1 + s_2$. Chaque nœud doit traiter une tâche de 80 bits, et le calcul peut être réécrit comme suit $kG = s_1 2^{80}G + s_2 G$. Comme le point $2^{80}G$ n'est pas précalculé et stocké en mémoire, avant de passer au traitement de la tâche, un des 2 nœuds doit d'abord passer 465 ms pour préparer le point. D'ailleurs, nous pouvons remarquer que l'algorithme de 2^xG n'est pas assez efficace, il lui faut 1293 pour préparer le point $2^{80}G$, ce n'est donc pas un choix adéquat pour notre méthode de parallélisme.

Nombre de nœuds :	1	2	3	4	5
Temps de calcul :	1003.55	1011.04	719.93	659.51	491.26
Gain :		-0.75%	28.26%	34.28%	51.05 %

TABLE 4.12 – Temps de calcul (ms) avec le parallélisme sans points précalculés

Les temps de calcul utilisant le parallélisme sans points précalculés sont donnés dans le tableau 4.12. Les résultats ont montré que quand nous parallélisons le calcul entre 2 nœuds, il n'y a aucune accélération, le calcul a même été ralenti. Nous commençons à avoir un gain positif lorsque plus de 3 nœuds participent au calcul, si nous comparons avec les résultats dans le tableau 4.6, nous pourrons remarquer qu'il y a une baisse de performance de presque 50%.

4.7/ CONCLUSION

Pour conclure la deuxième partie du test, les résultats ont montré que les algorithmes de 2^xG et de 2^2G ne sont pas aussi efficaces que prévus, notamment celui de 2^XG. En outre, l'algorithme de 2^2G est créé pour parcourir 2 bits à la fois, mais la vitesse de calcul n'est clairement pas doublée. Certes, le calcul parallèle avec cet algorithme peut accélérer le calcul des multiplications scalaires sans stocker les points précalculés, il offre un gain positif lorsque plus de 2 nœuds sont impliqués dans le calcul, mais la préparation en temps réel du point engendre une baisse de performance importante. Sachant que nous sacrifions l'énergie du réseau pour essayer d'atteindre une accélération la plus élevée possible, lors que le parallélisme sans points précalculés offre un gain beaucoup moins intéressant. Donc l'utilisation de cette méthode en pratique doit être soigneusement étudiée, nous conseillons de l'utiliser quand la consommation de mémoire est considérée comme un facteur important. Par exemple une application dans laquelle les nœuds n'ont pas assez de mémoire disponible pour stocker des points précalculés.

5

TOLÉRANCE AUX PANNES DANS LES RÉSEAUX DE CAPTEURS SANS FIL

Sommaire

La tolérance aux pannes est un ensemble de techniques qui sont conçues et mises en place pour pallier aux différents dysfonctionnements dans un système informatique. Concrètement, un mécanisme de tolérance aux pannes peut être représenté sous différentes formes, que ce soit un programme, un protocole ou un composant matériel, il permet au système de continuer à fonctionner même en présence d'anomalies.

Suite au développement des technologies informatiques, d'une part, les systèmes deviennent de plus en plus puissants pour offrir de meilleurs services aux utilisateurs. D'autre part, la complexité des fonctionnalités fournies rendent en même temps les systèmes plus fragiles face aux différentes pannes inattendues. Parfois un système sans protection risque d'être complètement suspendu à cause d'une petite erreur pendant son fonctionnement. Ainsi, aujourd'hui la tolérance aux pannes est devenue une partie incontournable durant la conception de tout système informatique moderne.

Par rapport aux systèmes qui fonctionnement totalement en local, la tolérance aux pannes dans des architectures distribuées est un problème beaucoup plus difficile. Notamment dans les réseaux de capteurs, qui sont constitués d'un grand nombre de nœuds

ayant une puissance de calcul faible et des ressources très limitées. D'ailleurs, dans un tel réseau, toute échange de données doit passer par des connexions sans fil, qui ne sont généralement pas suffisamment fiables et souffrent des interférences et des attaques extérieures.

Des mécanismes traditionnels de tolérance aux pannes ne sont pas directement applicables aux réseaux de capteurs. Comme nous l'avons présenté précédemment, les nœuds ne sont pas assez puissants pour gérer des algorithmes et des protocoles très complexes. En outre, les réseaux de capteurs sont initialement conçus pour être déployés dans des zones où l'intervention humaine n'est pas toujours possible, et il n'y a aucun intérêt à construire un tel réseau, s'il n'est pas assez robuste pour endurer des environnements inconnus et potentiellement hostiles. Il est donc important de développer spécifiquement de nouvelles techniques moins coûteuses pour assurer la fiabilité des réseaux de capteurs.

Une autre raison pour laquelle la tolérance aux pannes est une technique indispensable est que les réseaux de capteurs sans fil sont un domaine de recherche et d'ingénierie relativement récent. Certaines applications développées pour ce domaine ne sont pas encore matures et stables, et beaucoup de problèmes sont encore en discussion pour trouver la meilleure solution [46].

Dans ce chapitre, nous présentons d'abord les objectifs de la tolérance aux pannes et les sources possibles des pannes, ensuite nous donnons un état de l'art de l'ensemble de techniques existantes qui nous permettent de détecter des pannes et de restaurer le bon fonctionnement du système dans les réseaux de capteurs.

5.1/ OBJECTIF DE LA TOLÉRANCE AUX PANNES

L'objectif de la tolérance aux pannes est constitué principalement de 2 parties, qui est respectivement la détection de pannes et la restauration de fonctionnalité. Avant de donner des explications détaillées, il faut d'abord comprendre la différence entre *faute*, *erreur* et *faillite* [105]. En effet, les pannes dans les réseaux de capteurs peuvent être classifiées en 3 niveaux :

– Faute : il s'agit d'une sorte de dysfonctionnement matériel ou logiciel qui peut causer des erreurs, par exemple, un défaut matériel ou un bogue de programme.
– Erreur : une erreur représente un état anormal du système qui peut éventuellement conduire le système à une faillite totale.
– Faillite : la manifestation des erreurs qui se produit lorsque le système a dévié de sa spécification et ne peut plus livrer des fonctionnalités prévues.

5.1.1/ DÉTECTION DE FAUTES

Théoriquement, il est impossible d'éviter complètement des fautes, et les anomalies qui surviennent durant le fonctionnement du réseau ne sont pas toujours prévisibles. Par contre, il est possible de capturer un état erroné du système et de faire le nécessaire pour qu'il puisse continuer à fonctionner, même d'une manière réduite.

Après le déploiement et le mise en service d'un réseau, les nœuds doivent effectuer pé-

riodiquement des opérations diagnostiques pendant la transmission et traitement de données. Lorsqu'une erreur est détectée, ils la signalent immédiatement aux autres nœuds qui s'en occupent pour réagir le plus vite possible.

Plusieurs techniques sont proposées pour gérer efficacement la procédure de diagnostic, une présentation plus détaillées sur la détection d'erreurs est donnée dans la section 5.3.

5.1.2/ RESTAURATION DE FONCTIONNALITÉS

Une fois l'erreur détectée et localisée, il faut que le réseau réagisse rapidement, il peut soit isoler la partie en panne, soit essayer de la réparer. Le but est de laisser le réseau continuer de fonctionner. Une méthode qui est largement utilisée est la redondance. C'est-à-dire que les composants qui sont essentiels pour le bon fonctionnement du système sont répliqués pour augmenter sa fiabilité en cas de pannes sévères. Dans un réseau de capteur, les composants à répliquer sont souvent des nœuds assurant des services particuliers, par exemple les nœuds passerelles dans des protocoles de routage.

Les nœuds ne sont pas capables de gérer des mécanismes très complexes, c'est aussi la raison pour laquelle les nœuds sont censés être déployés en masse, car nous pouvons souvent trouver un grand nombre de nœuds qui sont disponibles. Nous pouvons les utiliser pour soit substituer les nœuds en panne, soit assurer simultanément une seule fonctionnalité.

5.2/ SOURCES DE PANNES

Nous donnons dans cette section une liste de sources principales des différentes pannes dans les réseaux de capteurs [70] avec une expérimentation réalisée sur capteurs SAW.

5.2.1/ NŒUD DÉFECTUEUX

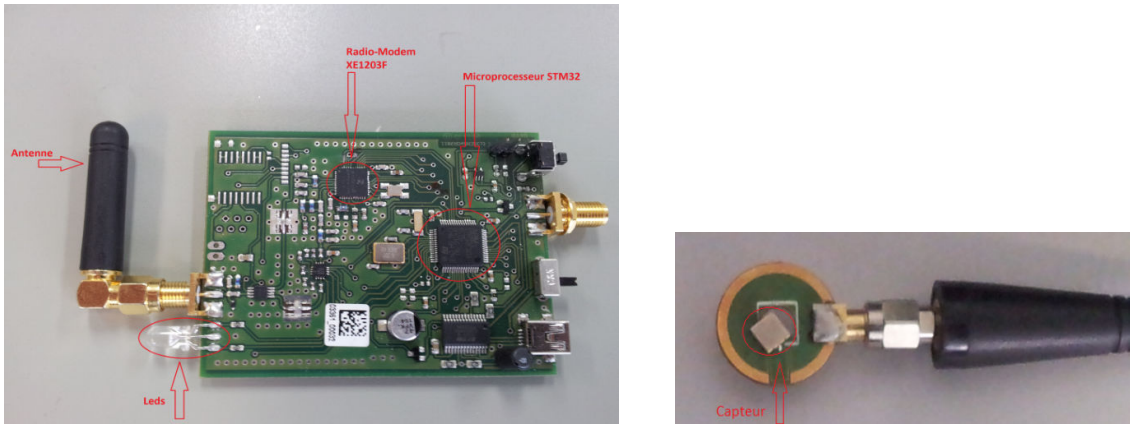
Un nœud est composé de différents composants matériels et piloté par des logiciels, qui subit éventuellement des pannes tout au long de son fonctionnement. Par exemple le boitier du nœud risque d'être cassé suite à des impacts physiques, qui peuvent endommager des matériels encapsulés dedans [64, 100, 98]. Dans [50], durant le déploiement des nœuds dans un champs de pommes de terre, les chercheurs ont trouvé que les antennes des nœuds sont tellement fragiles qu'elles émettent de temps en temps des signaux erronés. En outre, lorsque le niveau de pile d'un nœud devient faible, il y a des chances que ses composants ne puissent plus fonctionner correctement.

Dans [103], les nœuds sont déployés dans une forêt pour mesurer la température. Pendant les expérimentations, les chercheurs ont détecté beaucoup de données incohérentes. Certains nœuds ne produisent aucune donnée correcte pendant tout le test, les autres peuvent fonctionner au début du test, mais produisent aussi des données erronées avant l'épuisement de pile. On peut donc en déduire que la génération des données erronées est fortement corrélée à l'épuisement de la pile. Dans cette application, les chercheurs ont trouvé que lorsque le voltage de la pile n'est pas compris entre 3 et 2.4 volts,

le nœud commence à produire des données incohérentes. Certains nœuds envoient des températures de plus de 150 degrés quand le voltage est en dessous de 2.4 volts.

Dans notre projet européen INTERREG MainPreSi [68], on a installé des capteurs dans une machine pour aider l'entreprise TORNOS à mieux gérer son outil de production. On a utilisé 2 types de dispositif (voir figure 5.1) :

– Capteur SAW [1] qui fonctionne d'une manière passive, sans alimentation, et il peut être interrogé par des ondes radiofréquence à distance pour mesurer la température, la pression et les vibrations.
– Interrogateur qui émet des requêtes aux capteurs passifs et reçoit des données. Il est équipé d'un micro-contrôleur STM32, et il communique avec le capteur passif et les autres interrogateurs par la technique de multi-saut.



Interrogateur Capteur SAW

FIGURE 5.1 – Dispositifs utilisés dans le projet MainPreSi

Pendant les tests, on a constaté aussi des valeurs de température aberrantes. Après l'analyse des données générées, on a trouvé que les données erronées sont dues à une irrégularité des puissances des ondes qui interrogent les capteurs SAW. Pour que le capteur lise correctement la température, il faut que $P_{RX} > 800$ pour les capteurs mobiles, et $P_{TX} \leq 31 \wedge P_{RX} = 3000$ pour les capteurs fixes, où P_{TX} représente la puissance d'émission et P_{RX} est celle de réception.

Des pannes matérielles peuvent causer aussi des anomalies du logiciel installé. Par exemple, dans une application de collection de données, le programme ne peut pas effectuer correctement des traitements, si les composants de capture ne renvoient pas de données correctes. Une autre possibilité est le bogue, si le programme utilisé n'est pas strictement débogué avant l'installation, et il est installé sur les nœuds avec des bogues cachés, il y a des chances que le réseau tombe en panne bout d'un certain temps.

5.2.2/ PERTURBATION DE RÉSEAUX

Le routage est une fonctionnalité essentielle des réseaux de capteurs, c'est une technique fondamentale dont nous avons besoin pour la transmission de données. Des dys-

1. Surface Acoustic Wave

fonctionnements au niveau de routage peuvent engendrer des pertes et le retard de messages.

Généralement, dans un réseau de capteurs, les connexions entre les nœuds ne sont pas considérées comme très fiables, et le taux de délivrance des messages varie fortement selon les conditions ambiantes. L'expérimentation effectuée dans un champ agricole [101] montre aussi qu'il y a une baisse du taux de délivrance lorsque les plantes commencent à pousser.

Dans [97], les chercheurs ont construit un réseau de capteurs pour surveiller l'habitat des goélands. L'expérimentation a duré 115 jours avec 650,000 lectures effectuées par 98 nœuds déployés. Les fichiers de log ont montré un taux de délivrance des paquets très faible et une instabilité des connexions sans fil. Pendant le premier jour, les nœuds qui se situaient à un saut de la station de base ont assuré un taux de délivrance de 70%, alors que celui des nœuds à multi-sauts était seulement 58%. Pendant les jours restants, presque un quart des nœuds ont fonctionné avec un taux de délivrance de 35%. Le protocole de routage utilisé est considéré comme la cause principale de cette inefficacité du réseau, car il demande aux nœuds de choisir toujours le chemin le plus sûr. De ce fait, 80% des paquets sont passés par les 20% des chemins disponibles. Les piles des nœuds qui se situe sur ces chemins ont été épuisées au bout d'un jour et demi.

Il existe encore beaucoup d'autres causes possibles, par exemple la collision, comme les nœuds partagent le même médium pour la transmission de données, des collisions peuvent se produire lorsque plusieurs nœuds essayent d'émettre des signaux en même temps [98].

5.2.3/ DYSFONCTIONNEMENT DE LA STATION DE BASE

Toujours dans une application de collection de données, afin de faciliter et fluidifier la communication radio, l'ensemble d'informations collectées est agrégée au fur et à mesure durant leur transmission. Elles se propagent d'un nœud à l'autre pour aller jusqu'à leur destination finale, la station de base. Cette dernière peut aussi être victime des attaques extérieures et vulnérable aux différentes pannes.

Si aucun mécanisme de protection n'est appliqué, le réseau risque d'être complètement isolé. La station de base est la seule interface entre les nœuds et les utilisateurs, lorsqu'elle tombe en panne, les utilisateurs ne pourront plus accéder aux services fournis par le réseau, et dans l'autre sens, les nœuds ne pourront plus à envoyer données ou recevoir des commandes des utilisateurs.

Comme celles des nœuds déployés à distance, les pannes au niveau de la station de base peuvent aussi être causées par des dysfonctionnements matériels, ou des bogues qui se cachent dans le programme exécuté.

5.3/ TECHNIQUES DE DÉTECTION DE PANNES

La détection de pannes est une procédure cruciale, sans laquelle le système ne peut pas choisir et prendre la réaction adaptée pour minimiser les impacts, et les services qu'il fournit risque d'être suspendus. Il s'agit généralement d'un diagnostic du fonctionnement d'un composant spécifique, parfois, nous essayons aussi d'avoir une prédiction de

pannes, c'est-à-dire une analyse des symptômes observés.

Dans un réseau de capteurs, les nœuds sont censés être déployés en masse, et une maintenance manuelle n'est pas intéressante du tout, car trop coûteuse et compliquée. Dans cette section, nous présentons un ensemble de techniques qui nous permettent d'automatiser la procédure de détection de pannes dans les réseaux de capteurs.

5.3.1/ DIAGNOSTIC LOCAL

Dans certains cas, un nœud est capable de détecter des erreurs en faisant un diagnostic local. Afin de tester les connexions avec ses voisins, une table de routage contenant les identifiants des voisins est crée en mémoire locale, le nœud diffuse périodiquement des paquets de test aux voisins et attend des réponses. Le nœud est soupçonné d'être isolé, si aucun voisin ne lui répond, ou ils répondent, mais avec une puissance de signal très faible.

Un autre paramètre important qui peut être testé en local est le niveau de pile. Un nœud peut prévoir des pannes en mesurant régulièrement le voltage de la pile [7, 83]. Il existe des algorithmes qui permettent d'estimer la durée de vie d'un nœuds en analysant la courbe de décharge de la pile et son taux de décharge, comme sur les téléphones mobiles, le système peut afficher à l'utilisateur le niveau de batterie actuel. Lorsqu'un nœud détecte que sa pile est bientôt épuisée, il peut diffuser un message d'avertissement à ses voisins qui effectuent ensuite un ensemble d'opérations nécessaires pour l'exclure du réseau.

Dans certains cas, les données aberrantes peuvent être corrigées immédiatement après la détection. Comme dans [103], on sait que le composant de capture fonctionne correctement si le voltage de pile $v \in [2.4, 3.0]$. On peut donc demander aux nœuds de lire des données en parallèle avec le voltage, si $v \notin [2.4, 3.0]$, la donnée sera éliminée et la lecture sera refaite. Idem dans [68], on a les intervalles des puissances P_{TX} et P_{RX}, dans lesquels les capteurs peuvent fonctionner correctement, durant la réception des données, celles qui sont mesurées avec des puissances inappropriées sont éliminées.

5.3.2/ DIAGNOSTIC EN GROUPE

Il est aussi possible de détecter des erreurs du système causées par des valeurs erronées, si les nœuds disposent d'une valeur de référence. Une application qui est largement utilisée est la collection de données, nous supposons que les nœuds qui se trouvent dans la même zone sont censés capturer des valeurs similaires. Si nous trouvons une valeur qui possède une grande différence par rapport aux autres, cela est souvent considéré comme un symptôme d'erreur. Des algorithmes sont proposés dans [23, 48], ils peuvent calculer la probabilité qu'un nœud soit défectueux en se basant sur les données collectées.

Dans [82], les auteurs proposent de détecter des erreurs en testant l'incohérence des données collectées. Par exemple nous déployons des nœuds dans une chambre pour mesure la température ambiante. Un nœud est soupçonné être en panne, s'il renvoie des valeurs extrêmement élevées, tandis que les autres retournent des valeurs similaires et normales.

Nous supposons qu'un ensemble de nœuds $S = \{s_1, s_2, \ldots, s_n\}$ est déployé sous forme d'un arbre (voir figure 5.2), et à chaque intervalle temporel Δk, un nœud s_i crée un vecteur x_k^i contenant des données capturées $x_k^i = \{v_{kj}^i : j = 1 \ldots d\}$. Après m lecture, le nœud crée une collection de données $X_i = \{x_k^i : k = 1 \ldots m\}$, qui est envoyée à son nœud parent s_j. Ce dernier fusionne X_i avec sa propre collection X_j, et continue à envoyer la nouvelle collection à son parent. Ainsi, la collection remonte dans l'arbre d'un nœud à l'autre jusqu'à la station de base.

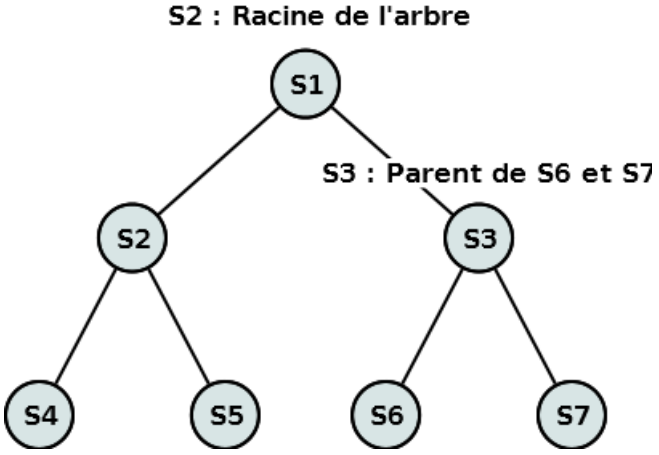


FIGURE 5.2 – Exemple de la topologie d'arbre

Deux approches sont proposées pour la détection d'erreurs. Dans le cas de détection centralisée, après la réception de la collection, la station de base applique un algorithme de clustering [25] à l'ensemble de données reçues. L'objectif est de représenter les données par des points dans un système de coordonnées cartésiennes, et de regrouper les points dans des clusters. 2 points sont placés dans le même cluster, si la distance euclidienne entre eux est inférieure au seuil choisi. Les points qui se situent en dehors des clusters représentent des données probablement corrompues, et elles sont donc ignorées. En traçant l'origine de ces données, la station de base peut identifier les nœuds défectueux.

L'inconvénient de cette approche est le coût de transmission, car il faut mobiliser entièrement le réseau pour effectuer un diagnostic. La 2e approche utilise une détection distribuée, c'est-à-dire l'algorithme de clustering est appliqué au fur et à mesure durant la transmission des données. Cette méthode nous permet de détecter des erreurs plus vite, et comme les données erronées sont ignorées après le clustering, la quantité de données à transmettre peut être considérablement réduite.

Dans certains cas, les nœuds sont programmés pour renvoyer des valeurs identiques, par exemple, les nœuds qui sont affectés délibérément à la même tâche. La récupération des résultats peut être considérée comme une sorte de vote, si un nœud renvoie une valeur qui est différente des autres, alors soit la valeur est corrompue durant la transmission, soit le nœud est défectueux.

Il est aussi possible de détecter des pannes en surveillance les réactions des nœuds [63]. Si les nœuds sont programmés pour recevoir des requêtes des autres et fournir des services demandés, nous pouvons vérifier si les requêtes sont correctement reçues par un nœud spécifique en observant son comportement. Si le nœud en question ne réagit pas comme il faut, soit il est défectueux, soit la connexion entre les 2 nœuds est perturbée.

5.3.3/ DIAGNOSTIC HYBRIDE

C'est en effet la combinaison des 2 modes de diagnostic présentées ci-dessus. C'est-à-dire que les nœuds effectuent d'abord chacun un diagnostic local, mais il y a toujours des chances que les résultats ne soient pas assez précis. Ensuite les nœuds échangent les résultats du diagnostic local pour faire un diagnostic de groupe.

Dans [15], Chen et al. proposent d'utiliser en même temps les diagnostics locaux et en groupe. Les nœuds sont déployés aléatoirement dans une zone, et nous supposons que chaque nœud possède au moins 3 voisins. Généralement, un réseau de capteurs contient un grand nombre de nœuds, et cette condition peut souvent être satisfaite facilement.

Après le déploiement d'un ensemble de nœuds S, et $|S| = n$, un nœud quelconque S_i dispose d'un certain nombre de voisins $N(S_i)$, et $|N(S_i)| = k$. Le nœud est programmé pour collecter périodiquement des données ambiantes x_i, et il peut comparer ses données collectées avec celles de ses voisins. d_{ij}^t représente la différente d'une donnée collectée au moment t entre S_i et son voisin S_j, et $\Delta d_{ij}^{\Delta t}$ signifie la différence d'une donnée entre S_i et S_j pendant la période Δt. Un paramètre $T_i \in \{LG, LF, GD, FT\}$ est utilisé pour représenter l'état de fonctionnement d'un nœud, où les valeurs signifient respectivement probablement bon, probablement défectueux, bon et défectueux.

Algorithme 20 : Algorithme de diagnostic local

```
1  pour tous les S_i ∈ S faire
2      pour tous les S_j ∈ N(S_i) faire
3          c_ij ← 0;
4          Calculer d_ij^t;
5          si |d_ij^t| > θ_1 alors
6              Calculer Δd_ij^Δt;
7              si |Δd_ij^Δt| > θ_2 alors
8                  c_ij ← 1;
9              fin
10         fin
11     fin
12 fin
```

Lorsque le diagnostic est lancé, dans un premier temps, les nœuds appliquent localement l'algorithme 20, où θ_1 et θ_2 sont 2 seuils prédéfinis, et $c_{ij} \in \{0, 1\}$ est le résultat du diagnostic, $c_{ij} = 0$ si les états de fonctionnements des 2 nœuds sont identiques, sinon $c_{ij} = 1$. L'idée de l'algorithme est toujours basée sur la supposition que les nœuds qui sont près l'un de l'autre sont censés avoir des données similaires. Si la différence de donnée dépasse θ_1 (ou θ_2 pour une période Δt), les 2 nœuds ont des états de fonctionnements différents, par exemple, un des 2 nœuds est défectueux.

Après le diagnostic local, le nœud essaie de déterminer son état possible à l'aide de l'algorithme 21 qui analyse les résultats c_{ij}. Le nœud S_i est probablement en bon état, si son état de fonctionnement est identique que ceux de la plupart de voisins.

Pour rendre le résultat de diagnostic plus précis, il faut que le nœud S_i passe ensuite au diagnostic en groupe (voir algorithme 22). Si S_i et plus de $3/4$ de ses voisins possèdent

Algorithme 21 : Détermination de l'état de fonctionnement possible

si $\sum_{S_j \in N(S_i)} c_{ij} < \lceil |N(S_i)|/2 \rceil$ **alors**
  $T_i \leftarrow LG$;
sinon
  $T_i \leftarrow LF$;
fin
Diffuser T_i aux voisins;

Algorithme 22 : Algorithme de diagnostic mutuel

si $\sum_{S_j \in N(S_i) \text{ et } T_j = LG} (1 - 2c_{ij}) \geq \lceil |N(S_i)|/2 \rceil$ **alors**
  $T_i \leftarrow GD$;
fin
Diffuser T_i aux voisins;

l'état de fonctionnement LG, alors S_i est considéré comme en bon état.

Algorithme 23 : Détermination de l'état de fonctionnement

pour $i \leftarrow 1$ **a** n **faire**
  si $T_i \in \{LG, LF\}$ **alors**
    si $T_j = GD, \forall S_j \in N(S_i)$ **alors**
      si $c_{ij} = 0$ **alors**
        $T_i \leftarrow GD$;
      sinon
        $T_i \leftarrow FT$;
      fin
    fin
  fin
fin

Ensuite pour les nœuds restant, nous utilisons l'algorithme 23 qui peut parcourir tous les nœuds. Si tous les voisins de S_i sont en bon état, et celui de S_i est identique que ceux de ses voisins, alors S_i est en bon état aussi, sinon, S_i est défectueux.

Cette approche divise la procédure de diagnostic en 2 parties, elle demande d'abord aux nœuds d'effectuer un diagnostic local qui renvoie un résultat représentant l'état de fonctionnement d'un nœud. Ensuite les nœuds échangent les résultats de diagnostic local avec leurs voisins, et en se basant sur les résultats reçus, ils peuvent calculer un deuxième résultat de diagnostic qui est plus précis.

Néanmoins, cette méthode ne permet pas d'effectuer un diagnostic partiel du réseau, car le résultat d'un nœud dépend tout le temps ceux de ses voisins, et pour tester les voisins, il faut mobiliser encore plus de nœuds. Ce genre de fonctionnement engendre une consommation de ressource assez importante, et la méthode n'est pas applicable dans les cas où nous voulons tester seulement quelques nœuds qui sont déployés dans une zone particulière.

5.3.4/ DIAGNOSTIC HIÉRARCHIQUE

La détection hiérarchique utilise la topologie du réseau pour la détection de pannes [90]. Nous supposons que les nœuds forment un arbre dans lequel chaque nœud peut avoir un ou plusieurs nœuds enfant, et chaque enfant ne possède qu'un seul nœud parent (voir figure 5.2).

Chaque nœud effectue périodiquement un diagnostic local qui génère un ensemble de données représentant son état de fonctionnement. Ensuite ces données sont envoyées directement à son nœud parent, qui les agrège en ajoutant ses propres données de diagnostic, et renvoie à son parent. Ainsi, les données remontent d'un nœud à l'autre, jusqu'à ce qu'elles arrivent à la station de base. A la fin de la procédure, la station de base dispose d'une base de données contenant toutes les informations nécessaires qui représentent l'état de fonctionnement de tout le réseau.

L'avantage d'un tel mécanisme est qu'il s'adapte parfaitement à la taille du réseau, quel que soit le nombre de nœuds déployés, la station de base peut toujours avoir une surveillance globale du réseau en diffusant une requête qui descend des nœuds parent aux nœuds enfant. Néanmoins, il faut mobiliser tous les nœuds pour effectuer un seul diagnostic, la transmission des données engendre une consommation de ressources très importante.

Une autre méthode alternative est d'utiliser seulement un sous-ensemble de nœuds et la station de base pour la détection de pannes [95]. Une fois les nœuds déployés, chacun envoie ses données de topologie (voisins, coût de connexion etc.) à la station de base qui peut construire chez elle la topologie entière de tout le réseau. Quand les nœuds détectent un nœud silencieux, c'est-à-dire un nœud qui ne répond plus aux requêtes, ils envoient des données nécessaires à la station de base pour que cette dernière puisse faire une mise à jour de topologie. Idem dans le sens descendant, si la station de base veut effectuer un changement de route, elle n'envoie la commande qu'aux nœuds concernés.

Cependant ce mécanisme n'est pas applicable aux applications événementielles dans lesquelles les nœuds ne communiquent qu'en cas de détection d'un événement important prédéfini, et la station de base ne peut pas avoir une vision globale du réseau en temps réel.

Une autre approche basée sur les clusters est présentée dans [91]. Chaque cluster-head est chargé de la surveillance de ses membres, et les cluster-heads sont directement surveillés par la station de base. Pour effectuer un diagnostic, la station de base et les cluster-heads effectuent des commandes ping vers les nœuds qui sont sous leur supervision directe. S'ils détectent un nœud qui ne répond pas à la commande ping, alors il sera considéré comme défectueux et sera éventuellement exclu du réseau.

5.4/ TECHNIQUES DE RESTAURATION

Les techniques de restauration sont souvent mises en place pour augmenter la fiabilité et la sûreté d'un système, car elles lui permettent de continuer à fonctionner malgré la présence des pannes. Beaucoup de méthodes sont proposées dans la littérature, alors que la plupart est basée sur la réplication des composants qui sont essentiels pour le

bon fonctionnement du système. La redondance peut être considérée comme une méthode adéquate pour les réseaux de capteurs, car les nœuds sont déployés en masse, et il existe souvent des nœuds disponibles pendant la durée de vie du réseau. Cependant comme de nombreux nœuds sont impliqués dans le fonctionnement du réseau, le système consomme plus de ressources. Dans cette section, nous présentons un ensemble de techniques de tolérance aux pannes qui sont basées sur la redondance.

5.4.1/ RÉPLICATION ACTIVE

L'idée est de déployer délibérément plusieurs nœuds pour assurer la même fonctionnalité. Par exemple, nous déployons plusieurs nœuds dans une zone pour effectuer une collection de données ambiantes. Lorsqu'un nœud tombe en panne, les autres qui se situent dans la même zone peuvent continuer à envoyer des données à la station de base.

5.4.1.1/ REDONDANCE DE CHEMIN DE ROUTAGE

Durant la conception d'un protocole de routage, il faut prendre en compte le fait que les nœuds qui se situent proche de la station de base sont relativement plus chargés pour la transmission de données, et leurs piles risquent d'être épuisées beaucoup plus vite par rapport aux autres. Pour que le réseau soit tolérant aux pannes, il ne faut pas que tout le réseau soit hors service à cause du dysfonctionnement de seulement quelques nœuds.

Dans un réseau tolérant aux pannes, les nœuds doivent être k-connecté, c'est-à-dire que chaque nœud doit posséder k connexions avec le reste du réseau, même s'il perd $k-1$ connexions, le nœud restera tout de même connecté [58].

[13] propose une topologie tolérante aux pannes pour les réseaux de capteurs hétérogènes. 2 types de nœuds sont utilisés, les nœuds de capture sont moins coûteux, mais très limités en ressources, ils sont déployés aléatoirement en grand nombre pour la collection des données ambiantes. Les nœuds de passerelle sont beaucoup moins nombreux, mais ils sont plus puissants, et leurs antennes peuvent avoir une portée plus étendue. Les nœuds de passerelle sont déployés pour récupérer des données collectées par des nœuds de capture, et transmettre ces données à la station de base. Une telle architecture est prouvée plus avantageuse en termes de performance et de durée de vie (voir figure 5.3).

Les nœuds de capture doivent être k-connectés, c'est-à-dire que chaque nœud de capture doit posséder k chemins distincts pour rejoindre un des nœuds de passerelle. Même un nœud de capture perd les autres $k-1$ connexions, il reste tout de même connecté avec les nœuds passerelles, et les données qu'il collecte peuvent toujours être renvoyées à la station de base. Par exemple, dans la figure 5.4, nous avons un réseau 3-connecté, dans lequel les nœuds de capture peuvent résister jusqu'à 2 chemins perturbés.

5.4.1.2/ REDONDANCE DE DONNÉES COLLECTÉES

Afin d'assurer la collection de données dans une zone, nous pouvons y déployer plusieurs nœuds, car les valeurs ambiantes capturées sont censées être similaires. Généralement

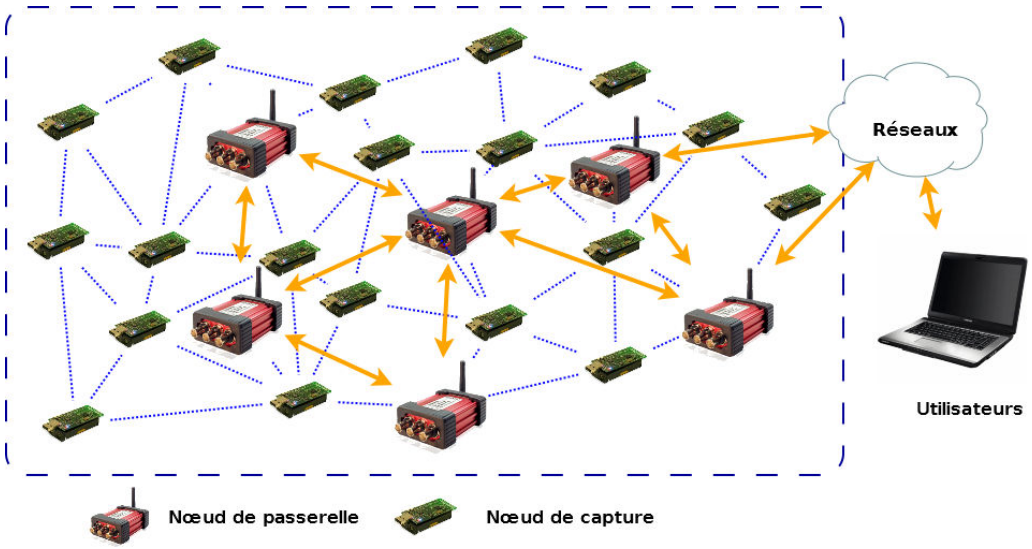


FIGURE 5.3 – Déploiement d'un réseau de capteurs hétérogène

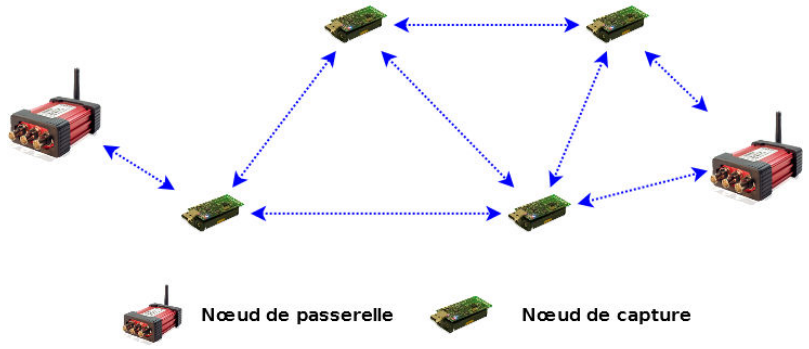


FIGURE 5.4 – Exemple d'un réseau hétérogène 3-connecté

les données capturées sont ensuite envoyées à un autre nœud qui est chargé de l'agrégation et de la transmission vers la station de base. Même si quelques nœuds tombent en panne, tant qu'il reste encore des nœuds opérationnels dans la zone, nous pouvons continuer à surveiller la zone malgré une perte de précision envisageable.

5.4.2/ RÉPLICATION PASSIVE

Quant à la réplication passive, nous mettons en place plusieurs nœuds aussi pour assurer la même fonctionnalité, mais seul le nœud principal est utilisé pour la réception et le traitement de tâches. Les données sont synchronisées entre le nœud principal et les nœuds de secours, c'est-à-dire que les nœuds de secours reçoivent aussi les tâches, mais ils restent inactifs tant que le nœud principal est encore opérationnel.

Lorsque le nœud principal tombe en panne, nous devons en choisir un parmi les nœuds de secours pour prendre le relais. La réplication passive est souvent considérée comme le choix optimal pour les réseaux de capteurs, car les nœuds de secours ne consomment pas beaucoup de ressource tant qu'aucune panne n'est détectée. L'application de la réplication passive est réalisée selon 3 phases, qui sont respectivement la détection de

pannes, la sélection du nœud principal et la distribution de service. La première phase est déjà discutée dans la section 5.3, et les 2 autres sont présentées ci-dessous.

5.4.2.1/ SÉLECTION DE NŒUD PRINCIPAL

Les nœuds sont programmés et déployés pour fournir des services, lorsqu'un nœud est en panne, il faut que nous trouvions un autre qui peut le remplacer, et le nœud sélectionné devient le nouveau fournisseur de services. Il existe plusieurs approches dans la littérature :

5.4.2.2/ SÉLECTION EN GROUPE

Comme la plupart des protocoles de routage basés sur le clustering [36, 35], le cluster-head est chargé de la récupération et la transmission des données collectées par les autre nœuds qui se situent dans le même cluster, et il est logique que le cluster-head consomme plus de ressources et sa pile se vide plus vite par rapport aux autres membres du cluster, notamment dans des réseaux homogènes, les cluster-heads ne détiennent pas plus de ressources que les autre membres. Lorsque le niveau d'énergie restante du cluster-head est très faible, il faut que les autres nœuds lancent un vote en prenant compte les paramètres cruciaux (énergie, distance, connexion etc.) pour choisir un nouveau cluster-head.

5.4.2.3/ RÉAFFECTATION DE MEMBRES

À partir d'un réseau de capteurs hétérogène dont les nœuds sont regroupés en clusters, et où les cluster-heads déployés sont plus puissants que les membres ordinaires, nous ne changeons pas de cluster-head. Chaque membre d'un cluster doit stocker non seulement l'identifiant de leur cluster-head actuel, mais aussi celui de son cluster-head de secours. Normalement le cluster-head de secours est le cluster-head d'un autre cluster qui se trouve à proximité. Quand le cluster-head actuel ne peut plus fonctionner, le membre envoie une demande à son cluster-head de secours pour être réaffecté au nouveau cluster [31].

5.4.3/ DISTRIBUTION DE SERVICES

Une fois le nouveau nœud sélectionné, il faut qu'il soit immédiatement activé, et qu'il détienne toutes les informations nécessaires pour continuer le traitement de tâches. Dans certaines applications, les nœuds déployés utilisent le même code, et les données sont synchronisées entre le nouveau et l'ancien nœud principal, il suffit alors de donner un simple message au nouveau nœud pour lui signaler le changement de rôle. Dans les autres cas, par exemple lorsque les nœuds n'ont pas assez de mémoire pour stocker le code de tout type de service, il faut donc que le code manquant soit distribué au nouveau nœud principal, et qu'il soit reprogrammé pour s'adapter à la nouvelle fonctionnalité.

5.4.3.1/ DISTRIBUTION DE CODES

Il s'agit de la dissémination des morceaux de code dans le réseau. Dans [56], un interpréteur de bytecode pour Tinyos est développé et installé sur des nœuds, et les programmes sont décomposés en morceaux de 24 instructions. Lorsqu'une mise à jour de code est demandée, les morceaux de code du nouveau programme sont envoyés et installés sur le nœud qui les exécute avec l'interpréteur. Cependant cette méthode demande une connexion assez fiable, car il faut que le programme à installer soit entièrement acheminé chez son destinataire. Le nœud ne peut pas se lancer tant que le programme utilisé n'est pas correctement mis à jour.

5.4.3.2/ DISTRIBUTION DE TÂCHES

Les programmes installés sur les nœuds sont identiques et seules les données représentant les tâches à effectuer sont envoyées au nouveau nœud sélectionné.

Les nœuds utilisent toujours le même programme qui peut être configuré en fonction de différents besoins. Pour modifier les fonctionnalités d'un nœud, il suffit de lui fournir l'ensemble des informations de configuration pour les nouvelle tâches.

Deluge [37] est une architecture spécifiquement conçue pour la dissémination de données. Les données sont représentées par des objets, qui sont sérialisés et divisés en pages de taille identique, et chacune est encore constituée d'un certain nombre de paquets (voir figure 5.5). Le paquet est l'unité la plus petite utilisée dans Deluge, et chacun est numéroté avec un nombre représentant la version des données qu'il contient.

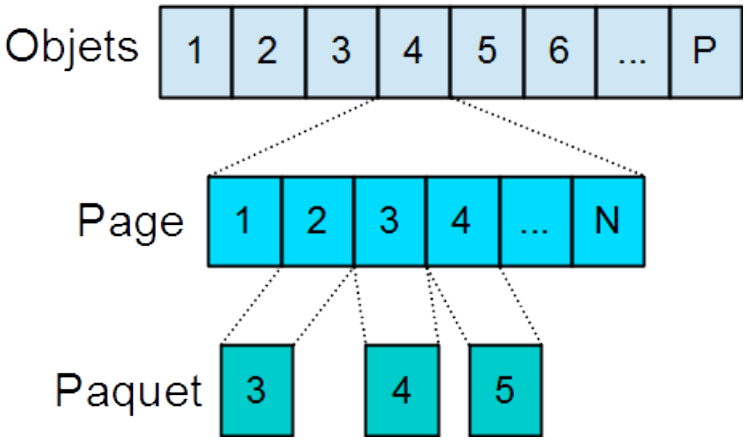


FIGURE 5.5 – Structures de données utilisées dans Deluge

Dans Deluge, après chaque mise à jour de données, un nœud S diffuse périodiquement aux voisins qui se situent dans sa portée radio la version la plus récente des données qu'il détient. Un voisin R qui reçoit le message lui demande une liste complète des objets avec les numéros de version correspondants. Avec cette liste, R peut identifier les objets locaux qui nécessitent aussi une mise à jour. Pour une raison de performance, les messages redondants sont rejetés pendant les communications. Par exemple si 2 nœuds se situent l'un à côté de l'autre, une seule demande de données est envoyée.

L'architecture elle-même utilise aussi des mécanismes qui lui offrent un minimum de sûreté. Après la réception d'une demande de données, le nœud S commence à envoyer à R

des paquets contenant des données demandées. Pour assurer la fiabilité de transmission de données, le contrôle de redondance cyclique (CRC) est appliqué à tous les paquets envoyés. Un paquet devra être renvoyé à nouveau, si le contrôle échoue. D'ailleurs, si le voisin perd la connexion durant la transmission de données, il va réessayer de demander les données manquantes. S'il n'arrive toujours pas à rétablir la connexion au bout d'un certain nombre d'essais, il va choisir à un autre nœud qui possède les données dont il a besoin.

5.5/ CONCLUSION

Dans ce chapitre nous avons présenté les techniques de tolérance aux pannes et leur application dans les réseaux de capteurs. A cause de la limitation des ressources et de la puissance de calcul, les réseaux de capteurs sont généralement très vulnérables aux différentes pannes, et parfois le réseau risque d'être entièrement bloqué à cause d'un simple dysfonctionnement. C'est la raison principale pour laquelle aujourd'hui les mécanismes de tolérance aux pannes sont considérés comme une partie indispensable pour les réseaux de capteurs.

Son objectif est principalement constitué de 2 parties, la détection des pannes et la restauration des fonctionnalités endommagées. Les algorithmes complexes sont à éviter, à cause des faibles ressources dont disposent les nœuds sans fil. Une approche efficace est d'utiliser les nœuds disponibles pour la détection de données erronées et la substitution de nœuds défectueux. Certaines valeurs peuvent être mesurées et évaluées localement, comme le niveau d'énergie restante, la puissance du signal des nœuds voisins etc. En effet, il ne faut pas que la méthode appliquée ait un impact négatif sur la performance du système.

Dans le chapitre suivant, nous présentons notre propre méthode de tolérance aux pannes conçue spécifiquement pour notre application de multiplication scalaire parallèle dans les réseaux de capteurs. Comme la tâche de calcul est décomposée en plusieurs parties qui sont traitées simultanément, il faut que la distribution de tâches et la récupération de résultats soient fiables, et que les nœuds soient capables de faire face aux pannes inattendues.

6

LA TOLÉRANCE AUX PANNES POUR LA MULTIPLICATION SCALAIRE PARALLÈLE DANS LES RÉSEAUX DE CAPTEURS

Sommaire

Dans le chapitre 4, nous avons présenté notre méthode de parallélisme qui nous permet d'accélérer le calcul des multiplications scalaires sur les courbes elliptiques en décomposant la tâche de calcul en plusieurs parties indépendantes qui sont ensuite traitées simultanément par des nœuds différents.

En tant qu'architecture de parallélisation sans mémoire partagée, les nœuds participant aux calculs parallèles doivent échanger des données via la connexion sans fil qui est souvent considérée comme un moyen de communication non fiable. S'il y a des données perdues ou corrompues pendant la transmission, le nœud maître ne va pas pouvoir terminer correctement le calcul. Le résultat du calcul est ensuite utilisé par les protocoles cryptographiques, et si les calculs sont basés sur des valeurs incorrectes, l'ensemble des mécanismes de sécurité du système va échouer.

Dans ce chapitre nous présentons des techniques qui nous permettent d'augmenter considérablement la fiabilité et la sûreté du réseau pendant le calcul parallèle. Comme nous l'avons dit précédemment, il est déconseillé de mettre en place des mécanismes de tolérance aux pannes très complexes dans les réseaux de capteurs, car les nœuds ne

sont pas assez puissants. Ainsi, notre méthode est principalement basée sur la redondance.

Nous commençons par l'identification des pannes possibles dans notre architecture de parallélisme. Afin de contourner ces problèmes, notre solution est présentée dans la section 6.2. Pour évaluer sa performance, nous avons développé un simulateur et les résultats de simulation sont présentés et discutés dans la section 6.3.

6.1/ ERREURS POSSIBLES DURANT LE CALCUL PARALLÈLE

En règle générale, un réseau de capteurs est constitué d'un grand nombre de nœuds à faible coût et puissance, qui sont souvent considérés très fragiles et vulnérables en face des différentes pannes et attaques [2, 112].

Il existe beaucoup de pannes différentes qui peuvent survenir durant le fonctionnement du réseau, et il est extrêmement difficile de développer une technique qui puisse résoudre tous les problèmes possibles. Généralement nous essayons d'abord d'identifier les pannes les plus importantes, c'est-à-dire les pannes qui peuvent avoir des effets désastreux sur notre système. Ensuite il faut trouver des méthodes qui peuvent renforcer notre réseau.

La procédure du calcul parallèle est présentée graphiquement dans la figure 4.9, nous pouvons voir que le nœud maître doit distribuer les tâches de calcul et récupérer le résultat de chaque esclave après le traitement de tâche. Tout échange de données se fait via la communication sans fil. Les erreurs qui peuvent avoir lieu durant le calcul sont listées ci-dessous :

6.1.1/ PERTE DE RÉSULTAT

Nous supposons qu'il y a n nœuds disponibles $s_i, i \in [1, 4]$ dans le cluster, ils sont convoqués en tant qu'esclaves par le maître, qui s'occupe de la génération et de la distribution des tâches Q_i qui est définie dans la formule 6.1, où l est la longueur du scalaire, n est le nombre d'esclaves. Chaque Q_i peut être ensuite traitée indépendamment par un esclave.

$$Q_i = \sum_{j=ib}^{i+b-1} k_j 2^j \cdot 2^b G \quad \text{où} \quad b = \lceil l/n \rceil \tag{6.1}$$

Il faut que le nœud maître reçoive tous les résultats des esclaves pour calculer le résultat final de la multiplication scalaire. Lorsqu'il y a des résultats perdus, le maître ne peut plus terminer son travail correctement. Une telle erreur peut être due aux différentes pannes ci-dessous :

– *Épuisement de pile* : Quand un des esclaves n'a plus d'énergie, il s'éteint automatiquement, et toute opération en cours s'arrête aussi.
– *Connexion perdue* : Une connexion sans fil devient beaucoup plus instable lorsque le niveau de pile du nœud est faible ou des interférences radio sont présentes dans la zone. Un résultat est perdu, si l'esclave correspondant ne reçoit pas de tâche, ou il n'arrive pas à renvoyer le résultat au maître à cause d'une mauvaise connexion.

– *Attaque extérieure* : Les nœuds déployés dans des environnements extrêmes font souvent l'objet des attaques délibérées ou accidentelles. Un nœud ne peut pas survivre à une telle attaque si ce petit dispositif électronique ne détient pas une protection solide.

6.1.2/ DONNÉES CORROMPUES

Comme tout message circulant dans un réseau de capteurs, les résultats renvoyés par des esclaves risquent d'être corrompus durant la transmission radio. Il existe des méthodes mathématiques qui peuvent tester l'intégrité d'un message, e.g. la fonction de hachage, mais généralement il est préférable de ne pas appliquer des calculs très compliqués qui peuvent parfois rendre les nœuds figés. En outre, l'objectif du calcul parallèle est d'accélérer le calcul de la multiplication scalaire, même si nous arrivons à intégrer le test d'intégrité dans le calcul parallèle, il peut y avoir une baisse de performance importante.

Si le maître ne peut pas détecter une telle erreur et continue à calculer le résultat final avec des valeurs incorrectes, les autres calculs suivants vont forcément échouer aussi. La corruption de données peut être principalement causée par :

– *Épuisement de pile* : Il est déjà prouvé que les composants électroniques ne peuvent pas fonctionner correctement lorsque le niveau de pile est très faible [12]. Si l'antenne du nœud n'est pas suffisamment alimentée, les signaux qu'elle émet risquent d'être biaisés.
– *Interférence radio* : Les interférences peuvent causer des difficultés de propagation des ondes radio, et parfois une mauvaise interprétation des signaux lors de la réception de données. Elle est souvent utilisée aussi comme un moyen d'attaque pour bloquer complètement la communication dans un réseau.
– *Bogue de programme* : Des résultats incorrects peuvent aussi être causés par des bogues du programme installé, qui peuvent être déclenchés par certaines conditions. C'est la raison pour laquelle les programmes doivent souvent passer plusieurs procédures de simulation stricte avant d'être installés sur des nœuds.

6.1.3/ NŒUD MAÎTRE DÉFECTUEUX

Dans notre architecture de calcul parallèle, c'est le cluster-head qui joue le rôle du maître. Il est chargé de la distribution des tâches et de la récupération de résultats. Un maître défectueux peut causer un abandon complet de la procédure du calcul parallèle.

Dans les protocoles de routage qui sont basés sur le clustering, les membres doivent choisir périodiquement un nouveau cluster-head [36, 35]. Ce mécanisme est particulièrement important pour les réseaux homogènes, dans lesquels les cluster-heads ne détiennent pas plus de ressources que les autres membres. Le changement périodique du maître permet aux nœuds d'équilibrer la charge et de prolonger la durée de vie du réseau.

Si un cluster-head tombe accidentellement en panne, et les autres membres vont détecter la perte de connexion avec leur cluster-head, et une procédure d'élection sera lancée immédiatement pour choisir un nouveau cluster-head.

Si le réseau est capable de détecter les erreurs ci-dessus et de restaurer les fonctionnalités, il sera plus robuste pour endurer des environnements difficiles et résister aux différentes pannes inattendues. Maintenant il faut trouver des méthodes efficaces qui nous permettent d'atteindre ces objectifs.

6.2/ CONTRE-MESURES CONTRE LES PANNES POSSIBLES

Selon [47], les techniques de tolérance aux pannes sont conçues et peuvent être appliquées (de basse à haute) aux couches physique, logiciel système, middleware et application. Généralement, plus la couche est basse, plus la technique de tolérance aux pannes est générique. Dans le sens inverse, plus la couche est haute, plus la technique est spécifique. Par exemple, il y a très peu de chance que nous puissions utiliser dans une application une technique qui est initialement développée pour une autre application.

Certes, les techniques qui sont développées pour la couche application sont moins portables et réutilisables par rapport aux autres. Cependant elles sont souvent soigneusement conçues pour mieux s'adapter aux besoins des applications. Notre architecture de parallélisme est une procédure complexe qui dépend beaucoup de la fiabilité de la communication sans fil. Afin d'augmenter la sûreté du système durant la distribution de tâches et la récupération de résultats, nous avons développé notre méthode de tolérance aux pannes pour la couche application.

6.2.1/ SÉLECTION DES NŒUDS DE SECOURS

Comme nous avons présenté précédemment, notre méthode est aussi basée sur la redondance, c'est-à-dire l'utilisation des nœuds de secours, car un réseau de capteurs possède souvent un très grand nombre de nœuds.

Nous supposons qu'il existe n nœuds disponibles dans le cluster. Lorsque le cluster-head veut lancer une multiplication scalaire parallèle, au lieu d'impliquer tous les nœuds disponibles dans le calcul, nous utilisons seulement $\lfloor n/2 \rfloor$ nœuds pour le calcul parallèle, les nœuds restants sont utilisés comme des nœuds de secours.

Algorithme 24 : Sélection des nœuds de secours

Données : L_a
Résultat : L_s, L_b

```
1 Trier L_a par le nombre de voisins dans l'ordre croissant;
2 pour tous les a ∈ L_a faire
3     b ← NS(a);
4     L_a ← L_a − a;
5     L_a ← L_a − b;
6     L_s ← L_s + a;
7     L_b ← L_b + b;
8 fin
9 retourner L_s, L_b
```

La sélection des nœuds esclaves et des nœuds de secours est pilotée par l'algorithme 24

où L_a est l'ensemble de nœuds disponibles dans le cluster, L_s et L_b sont 2 listes contenant respectivement des esclaves et des nœuds de secours sélectionnés.

Avant de commencer à sélectionner les nœuds, la liste L_a est triée dans l'ordre croissant par le nombre de voisins de chaque $a \in L_a$, car nous devons laisser le nœud ayant moins de voisins choisir son nœud de secours en premier. $NS(a)$ est une fonction qui sélectionne et retourne le nœud de secours de a parmi ses voisins. Les voisins sont évalués par plusieurs critères, telles que l'énergie restante, la distance de a et le RSSI [1], car nous privilégions toujours le voisin ayant la connexion la plus fiable.

Dans le chapitre 4, nous avons déjà montré que pour une raison de surcoût de communication, le nombre d'esclaves doit être limité à moins de 5. La possibilité de pouvoir effectuer une multiplication scalaire parallèle qui soit tolérante aux pannes dépend principalement du nombre de nœuds disponibles dans le cluster, dénoté n.

- $n \in [0, 1]$: Impossible d'effectuer une telle opération, car soit il n'y a pas de nœud disponible pour paralléliser le calcul, soit nous pouvons paralléliser entre 2 nœuds (maître compris), mais il n'y a pas de nœud de secours pour l'esclave.
- $n \in]1, 8[$: La tolérance aux pannes est possible, nous pouvons paralléliser le calcul avec $\lfloor n/2 \rfloor$ esclaves et $\lfloor n/2 \rfloor$ nœuds de secours.
- $n \in [8, \infty]$: Nous pouvons paralléliser le calcul avec 4 esclaves et au moins 4 nœuds de secours.

Après la sélection d'esclaves et de nœuds de secours, dans les sections suivantes, nous allons présenter comment les nœuds de secours sont mis en place pour rendre la procédure de calcul parallèle plus fiable.

6.2.2/ DÉTECTION DE PERTE DE RÉSULTAT

Pour détecter une perte de résultat, nous demandons à un nœud de secours de traiter la même tâche que l'esclave qu'il surveille, et de préparer une deuxième copie du résultat. La procédure du calcul est illustrée dans la figure 6.1. Le début est identique à celle du calcul parallèle normal (figure 4.9) :

- Les membres du cluster collectent et renvoient périodiquement des données au cluster-head ;
- Si le cluster-head détecte un événement important qu'il faut signaler immédiatement à la station de base, il essaie d'identifier les membres qui sont disponibles pour le calcul parallèle en leur diffusant une demande de participation ;
- Les membres qui sont disponibles renvoient une réponse positive au cluster-head, et attendent l'affectation des tâches.

Après la réception des réponses, le cluster-head utilise l'algorithme 24 qui lui sélectionne les esclaves pour le calcul parallèle et les nœuds de secours pour la tolérance aux pannes. Une fois le calcul décomposé en fonction du nombre d'esclaves, le cluster-head diffuse les tâches à tous les nœuds choisis.

La distribution des tâches se fait par broadcast qui est reçu par tous les membres disponibles du cluster. Dans le message, ils peuvent trouver les tâches de calcul et les identifiants des nœuds qui doivent les traiter, ainsi que les identifiants des nœuds qui

1. Received Signal Strengh Indicator

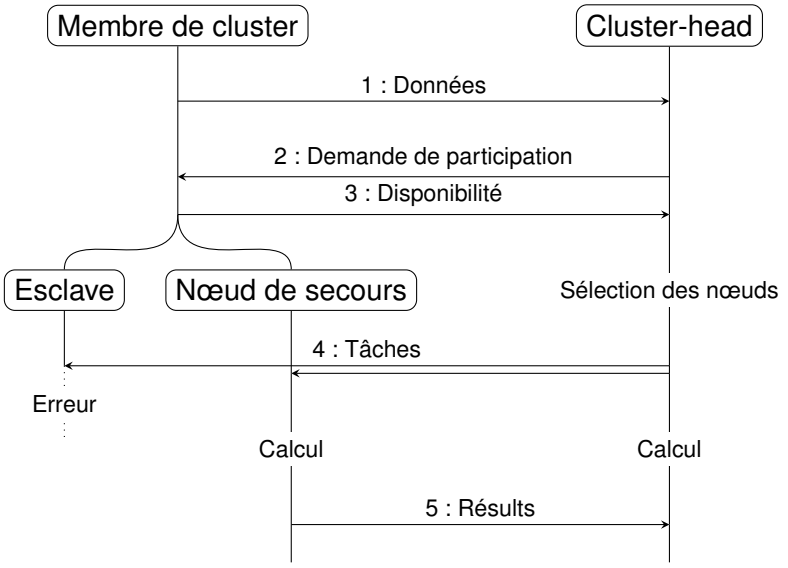


FIGURE 6.1 – Multiplication scalaire parallèle avec la détection de perte de résultat

doivent surveiller les esclaves. Après la distribution des tâches, les membres qui ne sont pas sélectionnés retournent à leur état initial.

Après le traitement de tâche, le nœud de secours attend que l'esclave qu'il surveille envoie le résultat du calcul. Si l'esclave tombe en panne, et n'envoie pas de résultat au bout d'un certains temps t_s, le nœud de secours va envoyer sa propre copie du résultat au maître.

Nous supposons que l'esclave et le nœud de secours ont presque la même distance du maître, car le nœud de secours est sélectionné parmi les voisins de l'esclave. La consommation d'énergie pour l'envoi de résultat est fonction de la taille du message m et de la distance d, notée $E_{T_x} = e(m, d)$, et la consommation pour la réception est une constante E_{R_x} [36, 35].

Nous pouvons constater que par rapport à la version originale, la nouvelle architecture parallèle avec la détection de perte de résultat n'engendre pas une augmentation importante de la consommation d'énergie du cluster. Néanmoins, la nouvelle version prend un peu plus de temps, car il faut que le nœud de secours attende t_s ms avant d'envoyer son résultat.

6.2.3/ VÉRIFICATION DE RÉSULTAT

Dans le cas où l'esclave renvoie un résultat incorrect au maître, il faut que les nœuds puissent le détecter. La procédure de vérification est représentée graphiquement dans la figure 6.2. Comme la détection de perte de résultat, après la distribution des tâches, l'esclave et le nœud de secours reçoivent et travaillent sur la même tâche.

Le nœud de secours est utilisé pour vérifier le résultat envoyé par l'esclave. Lorsque ce dernier envoie son résultat, comme le nœud de secours est un de ses voisins qui se situe dans sa portée radio, il peut le recevoir et passe à la phase de vérification.

Le nœud de secours compare le résultat de l'esclave avec le sien, car les données sto-

ckées localement sont considérées assez fiables. Si les 2 résultats sont différents, il enverra immédiatement un avertissement au maître, qui est obligé de rejeter le résultat en question et de répéter localement la tâche correspondante.

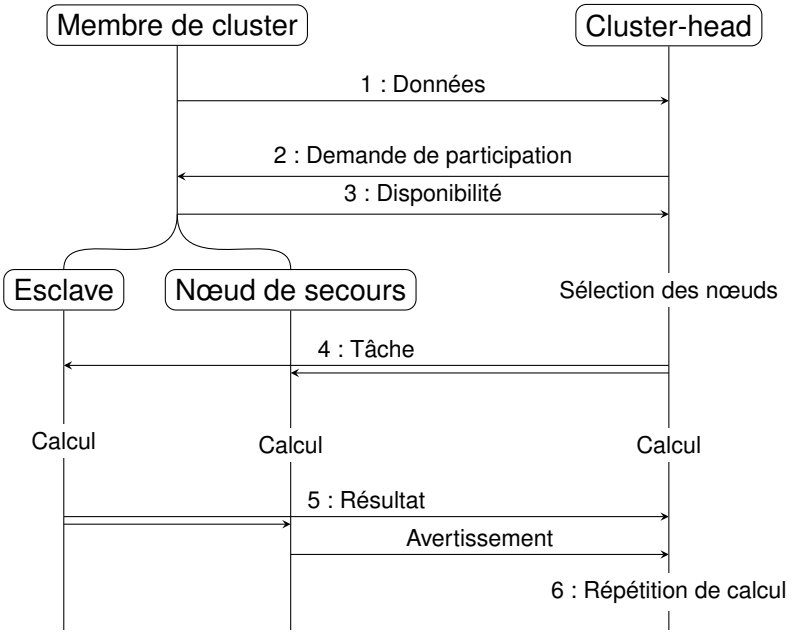


FIGURE 6.2 – Multiplication scalaire parallèle avec la vérification de résultat

Quand aucun résultat erroné n'est détecté, les nœuds de secours demeurent silencieux. Le temps de calcul et la consommation totale d'énergie restent quasiment inchangé. Cependant, lorsqu'il y a des esclaves qui retournent un résultat incorrect, le maître devra répéter la tâche avant d'effectuer la combinaison des résultats, et le coût de parallélisation est proportionnel au nombre de résultats erronés détectés.

Une autre méthode alternative plus économique en terme de consommation de ressources est d'utiliser plusieurs nœuds de secours pour chaque esclave et de mettre en place un mécanisme de vote. Lorsque les nœuds de secours détectent un résultat différent, ils envoient aussi leurs copies locales au maître, qui prend ensuite celui le plus présent.

Une limitation de la deuxième méthode est le nombre de nœuds utilisés. Si nous devons désigner plusieurs nœuds de secours pour un seul esclave, nous aurons besoin de beaucoup plus de nœuds disponibles dans le cluster. D'ailleurs, il faut que le nombre de nœuds de secours soit suffisamment grand pour que le résultat de vote soit assez fiable. De plus, comme les nœuds de secours doivent traiter les tâches aussi, il est évident que nous ne pouvons pas demander à un nœud de secours de surveiller plusieurs esclaves.

Pour le cas où le cluster-head tombe en panne, un mécanisme d'élection de nouveau cluster-head est déjà inclus dans la plupart de protocoles de routage basés sur le clustering. Il nous suffit de l'utiliser dans notre architecture de parallélisme.

6.3/ SIMULATION ET ÉVALUATION DE PERFORMANCE

Pour tester la performance de notre méthode de tolérance aux pannes, nous avons créé un simulateur en Java qui peut *estimer* le temps de calcul et la consommation d'énergie de notre architecture.

6.3.1/ SIMULATEUR ET CONFIGURATION

Au lancement du programme, le simulateur charge en mémoire la configuration de la courbe elliptique utilisée et la topologie du cluster depuis un fichier de configuration, dans lequel nous stockons les identifiants des membres du cluster, les distances entre eux, ainsi que l'ensemble de paramètres de la courbe, comme dans le tableau 4.5.

Nous supposons que les nœuds sont déployés dans une zone idéale sans obstacle. L'accès au médium est contrôlé par un protocole MAC basé le principe de TDMA [2], qui permet à un seul nœud d'envoyer des données à la fois.

Tous les membres du cluster utilisent la même courbe qui est définie dans un corps premier fini $\mathbb{F}_p$, et les paramètres (p, a, b, G, n) à stocker sont donnés dans le tableau 6.1, et les valeurs recommandées $NIST_{192}$ sont utilisées [34].

Paramètre	Description	Longueur (bit)
p	Taille du corps premier fini	192
a	Coefficient de l'équation Weierstrass simplifiée	8
b	Coefficient de l'équation Weierstrass simplifiée	192
G	Point de générateur de la courbe	$192 \times 2 = 384$
n	Ordre du point de générateur	192
	Total :	968

TABLE 6.1 – Paramètre de la courbe elliptique utilisée

Les détails de la consommation de mémoire sont montrés dans le tableau 6.2. Selon les résultats dans le chapitre 4, le nombre de nœuds participant au calcul parallèle doit être limité à moins de 5, nous devons stocker donc 3 points précalculés.

Données	Description	Taille (octet)
(p, a, b, G, n)	Paramètres de la courbe elliptique	121
$2^{40}G, 2^{80}G, 2^{120}G$	Points précalculés	144
	Total :	256

TABLE 6.2 – Consommation de mémoire pour le stockage des données

Comme dans un réseau de capteurs, la plupart de l'énergie est consommée par la communication sans fil. Ainsi, le simulateur prend en compte seulement la consommation d'énergie de la communication radio, celle du traitement de données en local est simplement ignorée.

Les formules de l'estimation de la consommation sont présentées dans [36, 35] (voir formule 6.2). E_{T_X} et E_{R_X} sont 2 fonctions qui estiment la consommation d'énergie d'émission

2. Time Division Multiple Access

et de réception, où d est la distance et k est la taille du message en nombre de bits. E_{elec} est le facteur de dissipation audiofréquence, et ϵ_{amp} est la facteur d'amplification de transmission. La modélisation de la consommation d'énergie de la transmission radio est en discussion, nous avons repris les valeurs données dans le papier : $E_{elec} = 50nJ/bit$ et $\epsilon_{amp} = 100pJ/bit/m^2$.

$$\begin{cases} E_{T_X}(k,d) &= E_{elec} \cdot k + \epsilon_{amp} \cdot k \cdot d^2 \\ E_{R_X}(k) &= E_{elec} \cdot k \end{cases} \tag{6.2}$$

Les nœuds communiquent entre eux avec le protocole Zigbee, qui est largement utilisé par la plupart des plate-formes existantes. Pour estimer le temps de communication, le simulateur utilise les valeurs de la bande passante présentées dans [11]. Les auteurs ont prouvé que selon la dernière version de la norme IEEE 802.15.4-2006, la bande passante du Zigbee est 108 Kbps.

De plus, l'estimation du temps de calcul de la multiplication scalaire parallèle est principalement basée sur les résultats du chapitre 4.

La mission du simulateur est d'*estimer* le temps de calcul et la consommation d'énergie de notre architecture de parallélisation protégée par les mécanismes de tolérance aux pannes que nous avons présenté dans la section 6.2. L'objectif de la simulation n'est pas de *mesurer* précisément les valeurs en question, mais d'étudier l'impact des techniques de tolérance aux pannes sur la performance du calcul parallèle.

Les tests sont exécutés dans 3 cas différents :
– Il existe assez de nœuds disponibles dans le cluster pour paralléliser les multiplications scalaires jusqu'à 4 esclaves, mais il n'y en a pas assez pour avoir un nœud de secours par esclave.
– Il existe assez de nœuds disponibles pour paralléliser le calcul entre 4 esclaves, et chaque esclave détient un nœud de secours.
– Il existe toujours assez de nœuds disponibles pour le calcul parallèle (esclave) et la tolérance aux pannes (nœud de secours), et certains esclaves tombent en panne pendant le calcul.

6.3.2/ TEST SANS ASSEZ DE NŒUDS DISPONIBLES ($n \leq 4$)

Dans le chapitre 4, nous avons déjà montré qu'à cause du surcoût de la communication radio, nous devons limiter le nombre d'esclaves à moins de 5. Nous définissons le nombre de nœuds disponibles dans le cluster $n, n \leq 4$, et les résultats de simulation du premier cas sont montrés dans les tableau 6.3 et 6.4. La comparaison de performance est aussi illustrée dans la figure 6.3.

Nombre de nœuds	Temps de calcul (ms)	Énergie (mJ)
0	2308.77	0.06639066
1	1158.88	0.17869498
2	777.75	0.27243610
3	588.81	0.34194090
4	476.75	0.41024986

TABLE 6.3 – Performance sans nœud de secours ($n \leq 4$)

Nombre de nœuds	Temps de calcul (ms)	Énergie (mJ)
0	2308.70	0.06639066
1	2308.77	0.06639066
2	1159.01	0.16637882
3	1159.01	0.16637882
4	778.01	0.24765018

TABLE 6.4 – Performance avec nœuds de secours ($n \leq 4$)

Nous pouvons constater que sans nœuds de secours, tous les nœuds disponibles peuvent participer au calcul, le temps de calcul diminue progressivement avec l'augmentation du nombre d'esclaves. La consommation totale d'énergie est proportionnelle au nombre d'esclaves, car plus le nombre d'esclaves est élevé, plus l'échange de données est fréquente.

Cependant, si nous utilisons les nœuds de secours, il n'y a que la moitié des nœuds qui peuvent participer au calcul parallèle, et l'accélération du calcul devient moins efficace. C'est la raison pour laquelle le temps de calcul ne diminue que lorsque le nombre de nœuds est pair. Par contre, la consommation d'énergie avec les nœuds de secours devient beaucoup moins importante.

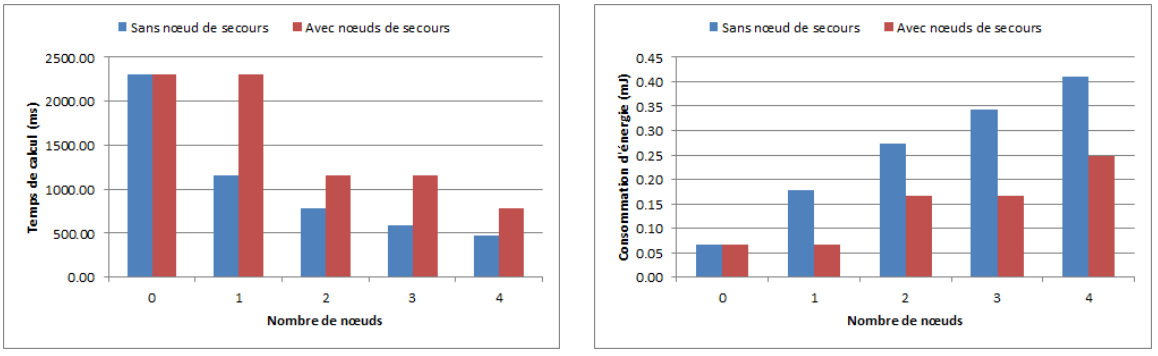


FIGURE 6.3 – Parallélisation sans assez de nœuds disponibles ($n \leq 4$)

Le nombre de nœuds utilisés est doublé lorsque notre méthode de tolérance aux pannes est appliquée. Ainsi, seulement $\lfloor n/2 \rfloor$ nœuds peuvent participer au calcul, et il nous faut plus de temps pour terminer le calcul. D'autre part, les nœuds de secours n'ont pas besoin de communiquer avec le maître quand il n'y a pas de panne, et la consommation totale d'énergie du cluster est moins élevée par rapport à celle du parallélisme sans nœud de secours.

6.3.3/ TEST AVEC ASSEZ DE NŒUDS DISPONIBLES ($n \leq 8$)

Dans le deuxième test, nous supposons que le cluster peut posséder jusqu'à 8 nœuds disponibles. Le cluster-head peut toujours paralléliser la multiplication scalaire entre 4 esclaves, peu importe si notre méthode de tolérance aux pannes est utilisée.

Comme nous pouvons voir dans le tableau 6.5 et la figure 6.4, le maître a besoin d'un peu plus de temps pour terminer complètement le calcul parallèle. Dans certains cas, il y a une petite augmentation du surcoût de communication radio, car chaque esclave est

Nombre d'esclaves	Temps de calcul (ms)	Énergie (mJ)
0	2308.70	0.06639066
1	1159.01	0.16637882
2	778.01	0.24765018
3	589.20	0.34410666
4	477.27	0.43921380

TABLE 6.5 – Performance avec suffisamment de nœuds de secours ($n \leq 8$)

surveillé par un nœud de secours, qui ont aussi besoin de recevoir des tâches du maître et des résultats des esclaves. D'ailleurs, après la réception des résultats, le maître doit attendre des avertissements éventuels avant de passer à la combinaison des résultats.

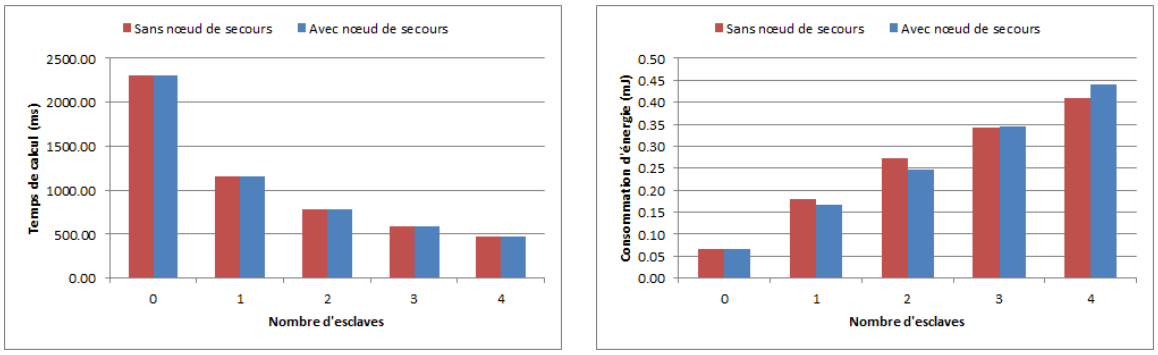


FIGURE 6.4 – Parallélisation avec assez de nœuds disponibles ($n \leq 8$)

D'un point de vue global, la consommation d'énergie des 2 types de parallélisation s'accroît proportionnellement avec l'augmentation du nombre d'esclaves. Car il existe suffisamment de nœuds disponibles dans le cluster qui nous permettent de paralléliser le calcul juqu'à 4 esclaves, même avec l'utilisation des nœuds de secours.

La différence de la consommation d'energie entre les 2 types de parallélisation est principalement due au déploiement aléatoire des nœuds dans le cluster, les distances entre les esclaves et les nœuds de secours ne sont pas toujours les mêmes.

6.3.4/ TEST EN PRÉSENCE DE PANNES

Pour le troisième test, nous supposons que nous avons toujours 4 esclaves opérationnels et chacun possède un nœud de secours. Pendant le calcul parallèle, si un esclave n'arrive plus à fonctionner correctement, il faudra que son nœud de secours puisse détecter l'erreur et essaie de restaurer le calcul.

6.3.4.1/ RÉSULTAT ERRONÉ

Lorsqu'un nœud de secours détecte que l'esclave qu'il surveille envoie un résultat erroné, il faut qu'il envoie immédiatement un avertissement au maître, qui va retraiter la tâche correspondante localement. En comparant la performance (temps de calcul et la consommation d'énergie) avec celle de la parallélisation sans détection de résultat erroné, nous pouvons constater dans le tableau 6.6 et la figure 6.5 que quand il y a des

résultats erronés, le cluster a besoin de beaucoup plus de temps et d'énergie pour terminer le calcul.

Nombre de résultats erronés	Temps de calcul (ms)	Énergie (mJ)
0	477.27	0.43921370
1	938.86	0.44161946
2	1400.44	0.44564062
3	1862.03	0.44965998
4	2323.61	0.45207022

TABLE 6.6 – Performance avec la détection de résultat erroné

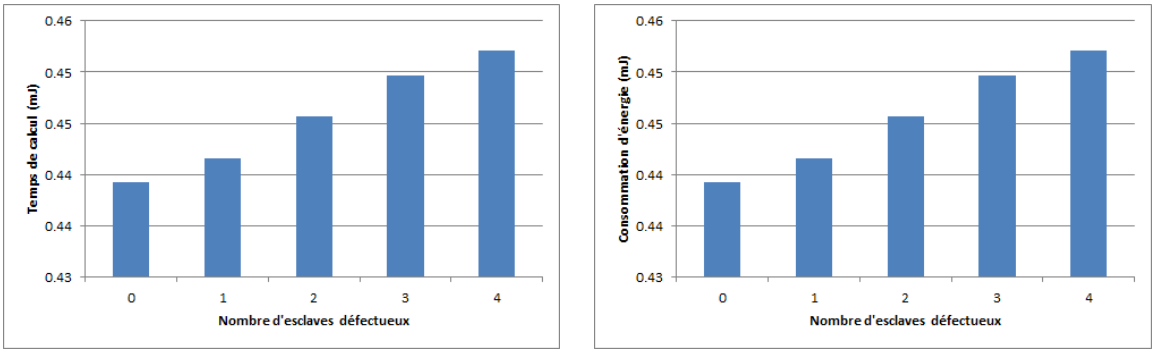


FIGURE 6.5 – Parallélisation avec la détection de résultat erroné

Nous pouvons voir aussi que le temps de calcul s'accroît très vite avec l'augmentation du nombre de résultats erronés. Quand il y a plus de 3 résultats erronés, le calcul parallèle ne donne quasiment plus d'accélération. Contrairement au temps de calcul, la consommation totale d'énergie augmente relativement moins rapidement, car la taille du message d'avertissement est très petite, les valeurs de la consommation d'énergie est toujours comprise entre 0.44 mJ et 0.45 mJ.

6.3.4.2/ RÉSULTAT PERDU

Un nœud de secours est aussi chargé de la détection de résultat perdu. Comme l'esclave qu'il surveille, le nœud de secours doit traiter aussi la même tâche que son esclave. Si l'esclave renvoie correctement son résultat au maître, celui du nœud de secours est simplement supprimé. Dans le cas contraire, si l'esclave ne renvoie pas de résultat dans un délais prédéfini, le nœud de secours doit envoyer sa copie locale du résultat au maître, pour que ce dernier puisse continuer son travail.

Nombre de résultats perdus	Temps de calcul (ms)	Énergie (mJ)
0	477.27	0.43921370
1	517.27	0.43859930
2	517.27	0.47736890
3	517.27	0.49638458
4	517.27	0.47632058

TABLE 6.7 – Performance avec la détection de résultat perdu

Dans le tableau 6.7, nous pouvons remarquer que le temps de calcul n'augmente plus quand il y a plus d'un résultat perdu. Car même il y a plusieurs résultats perdus, les nœuds de secours peuvent réagir simultanément, et le maître ne fait que prendre directement les résultats des nœuds de secours.

Quand plus d'un résultat sont perdus, le simulateur donne toujours exactement le temps de calul, car les résultats de simulation sont basés sur des formules et des paramètres théoriques. En réalité, les conditions environnementales sont plus complexes, il y a très peu de chance que les valeurs mesurées soient exactement les mêmes, mais normalement elles sont censées être similaires.

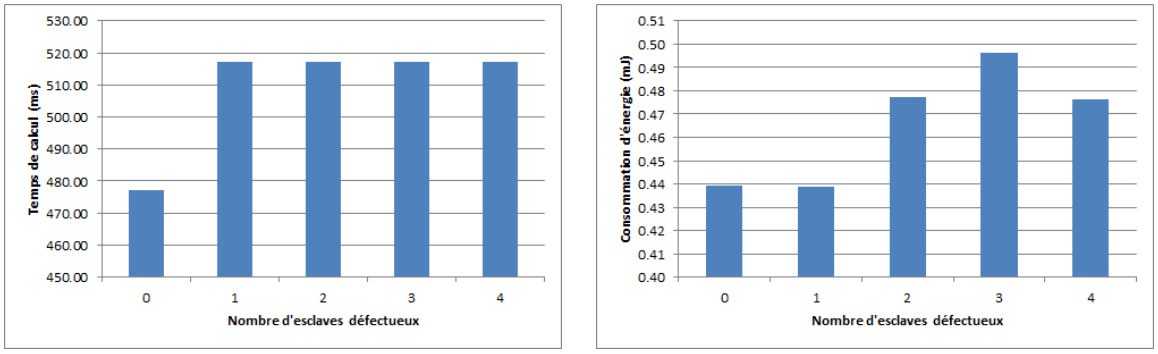


FIGURE 6.6 – Parallélisation avec la détection de résultat perdu

Dans la figure 6.6, la variation de la consommation d'énergie est très irrégulière, car la consommation d'énergie est calculée en fonction de la taille de message et la distance entre les 2 nœuds. Les nœuds sont déployés d'une manière aléatoire, les distances entre les nœuds de secours et le maître sont différentes.

6.4/ CONCLUSION

Après la simulation, nous pouvons voir que notre méthode de tolérance aux pannes a bien atteint son objectif initial, qui est composé de 2 parties : la détection de pannes et la restauration de fonctionnalités. Elle permet aux clusters de continuer à effectuer des multiplications scalaires parallèles en cas de pannes.

Lorsque aucune panne n'est détectée durant le calcul parallèle, notre méthode ne donne pas d'impact négatif important sur la performance du système. Cependant en cas de pannes, les nœuds de secours vont prendre le relais des esclaves défectueux et continuer à travailler sur les tâches qui leur sont affectées. Il faut qu'au final, le maître puisse calculer et obtenir le résultat de la multiplication scalaire, même en sacrifiant significativement la performance du système.

Les résultats de simulation ont montré qu'en ajoutant un nœud de secours par esclave, nous avons pu doubler la fiabilité du système. Néanmoins les opérations de restauration de fonctionnalité engendrent une augmentation de consommation importante. Pour prolonger la durée de vie du réseau, il faut bien gérer l'utilisation de parallélisme. Comme nous avons présenté dans le chapitre 4, le calcul parallèle ne doit être appliqué que dans les cas urgents. Ainsi, on va envoyer un message sûr, rapide et sécurisé. Ce type de système est principalement dédié aux applications critiques, comme les applications militaires ou de surveillance.

7

CONCLUSION

L'objectif de cette thèse était d'appliquer l'ECC dans les réseaux de capteurs sans fil d'une manière efficace et sûre. La difficulté principale est d'effectuer des calculs compliqués sur les nœuds qui disposent d'une puissance de calcul très limitée. Certes, ECC offre un calcul plus rapide et une utilisation de clés plus courtes par rapport à RSA, mais les calculs impliqués dans les protocoles cryptographiques restent toujours très complexes pour les micro-contrôleurs. Après une étude détaillée de la cryptographie sur les courbes elliptiques, nous avons constaté que l'opération la plus coûteuse est la multiplication scalaire. De nombreuses méthodes existent dans la littérature, mais elles ne sont pas suffisantes pour donner une accélération intéressante.

Une technique qui permet d'accélérer des calculs est le parallélisme, une tâche de calcul est découpée en plusieurs parties indépendantes, qui peuvent être traitées en même temps par des processeurs différents. Nous proposons donc de paralléliser les multiplications scalaires entre plusieurs nœuds qui se situent dans le même cluster, car les réseaux de capteurs qui sont basés sur le clustering sont prouvés plus efficaces en termes de consommation d'énergie.

En outre, la cryptographie symétrique peut offrir un calcul plus rapide, mais il n'est pas toujours facile de réaliser une distribution de clés à grande échelle dans un réseau de capteurs qui peut contenir un très grand nombre de nœuds. Nous proposons de limiter l'utilisation de la cryptographie symétrique à l'intérieur d'un cluster pour sécuriser la distribution des tâches, mais entre les clusters, nous appliquons la cryptographie asymétrique qui offre une gestion de clé plus sophistiquée.

Une spécificité importante des réseaux de capteurs est le déploiement massif et aléatoire des nœuds. Dans une application de surveillance, il peut exister plusieurs nœuds disponibles dans un cluster tant qu'aucun événement critique n'est détecté. Quand un événement important est capturé par le réseau, il est plus intéressant de demander à ces nœuds de coopérer collectivement pour générer un message bien chiffré le plus vite possible. Dans cette thèse, nos travaux de recherche se concentrent principalement sur la parallélisation des multiplications scalaires.

Comme les autres architectures de calcul parallèle, un inconvénient principal est la consommation d'énergie. Nous avons implémenté et testé notre solution sur les nœuds Telosb. Les résultats ont montré que nous pouvons obtenir un gain maximal de 74.8%. En effet, les nœuds ne disposent d'aucune mémoire partagée, et la distribution des tâches et la récupération des résultats sont obligées de passer par la communication radio, qui engendre une consommation d'énergie importante. D'ailleurs, à cause du surcoût de la communication, le nombre de nœuds participant au calcul doit être limité à moins de 5.

Nous concluons que la parallélisation peut réaliser une accélération intéressante du calcul des multiplications scalaires, mais cette technique risque de réduire considérablement la durée de vie des nœuds, car la communication radio génère une consommation d'énergie importante. Dans un réseau de capteurs, la parallélisation des multiplications scalaires ne doit pas être utilisée comme un paradigme de calcul par défaut. Elle peut être appliquée dans des applications de surveillance, dans lesquelles les nœuds demeurent inactifs pendant la plupart du temps pour préserver l'énergie. Le calcul parallèle ne peut être utilisé que dans les cas urgents où le réseau doit signaler à la station de base un événement important, par exemple la détection d'une catastrophe naturelle dans une application de surveillance environnementale.

Enfin la deuxième partie de la contribution traite de l'amélioration de notre solution de parallélisation. Nous proposons de mettre en place des mécanismes de tolérance aux pannes pour rendre la procédure du calcul parallèle plus fiable. L'application du parallélisme au sein d'une architecture sans mémoire partagée nécessite une échange de données via les connexions sans fil, qui sont souvent considérées comme un moyen de communication non fiable. Notre solution permet aux nœuds de détecter des résultats perdus ou erronés pendant le calcul parallèle, et de restaurer le calcul en cas de pannes. Notre simulateur a donné des résultats qui montrent que la solution appliquée peut rendre le calcul parallèle plus sûr sans avoir des impacts négatifs importants.

PERSPECTIVES

Pour l'instant, nos travaux de recherche se focalisent principalement sur la parallélisation des multiplications scalaires, et l'implémentation que nous avons utilisé pendant le test de performance peut être plus complète. En effet, elle permet de tester seulement la performance des multiplications scalaire parallèles, mais pas celle de l'exécution d'un protocole cryptographique complet. C'est pourquoi nous avons décidé d'essayer d'appliquer notre solution de parallélisation dans le projet qui est présenté dans l'annexe 7. Cependant le développement n'est pas encore terminé, nous allons donc continuer à travailler sur ce projet pour obtenir des résultats plus significatifs.

Par ailleurs, dans notre implémentation actuelle, nous avons déjà appliqué certaines méthodes mathématiques permettant d'améliorer la performance de la multiplication scalaire, comme la représentation NAF du scalaire et l'utilisation des coordonnées jacobiennes. Cependant, nous n'avons pas encore découvert tous les aspects de la courbe elliptique, il reste encore d'autres techniques à voir et tester, par exemple la courbe Montgomery, la courbe Koblitz, la méthode de la fenêtre. Notre technique de parallélisation est basée sur le découpage du scalaire, qui peut théoriquement coexister avec les autres méthodes, qui cherchent plutôt à modifier la représentation du scalaire ou la forme de la courbe. Il sera plus intéressant de combiner le parallélisme avec d'autres techniques, et de tester leur performance.

Un autre problème de notre solution de parallélisation est la consommation d'énergie, qui est largement reconnue comme un facteur critique pour les réseaux de capteurs, mais elle est aussi considérée comme un inconvénient inhérent du parallélisme, car plus de processeurs sont impliqués dans le traitement des tâches. Cependant, nous allons essayer de réduire la consommation d'énergie, qui peut influencer considérablement la durée de vie des réseaux.

LISTE DES PUBLICATIONS

1. Shou, Y. & Guyennet, H. **A Fault Tolerant Parallel Computing Scheme for Scalar Multiplication for Wireless Sensor Networks**. *Distributed Computing and Networking, Springer LNCS*, **2014**, 8314, 317-331.
2. Shou, Y. ; Guyennet, H. & Lehsaini, M. **Parallel Scalar Multiplication on Elliptic Curves in Wireless Sensor Networks**. *Distributed Computing and Networking, Springer LNCS*, **2013**, 7730, 300-314.
3. Faye, Y. ; Guyennet, H. ; Niang, I. & Shou, Y. **Fast Scalar Multiplication on Elliptic Curve Cryptography in Selected Intervals Suitable for Wireless Sensor Networks**. *Cyberspace Safety and Security, Springer LNCS*, **2013**, 8300, 171-182.
4. Faye, Y. ; Guyennet, H. ; Niang, I. & Shou, Y. **Optimisation d'un Protocole d'Authentification dans les Réseaux de Capteurs Sans Fil**. *SAR-SSI 2012, 7ème Conf. sur la sécurité des architectures réseaux et systèmes d'information*, **2012**.
5. Aupet, J.-B. ; Lapayre, J.-C. & Shou, Y. **Adaptabilité des flux vidéo dans les logiciels collaboratifs : Test de qualité par référence prédéfinie**. *Ubimob 2011, 7es journée francophones Mobilité et Ubiquité*, **2011**, 96-108.

BIBLIOGRAPHIE

[1] S Adve, Vikram S Adve, Gul Agha, Matthew I Frank, M Garzarán, J Hart, W Hwu, R Johnson, L Kale, R Kumar, et al. Parallel computing research at illinois : The upcrc agenda. *Urbana, IL : Univ. Illinois Urbana-Champaign*, 2008.

[2] Ian F Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. Wireless sensor networks : a survey. *Computer networks*, 38(4) :393–422, 2002.

[3] ZigBee Alliance. Zigbee specification, 2006.

[4] Bijan Ansari and Huapeng Wu. Parallel scalar multiplication for elliptic curve cryptosystems. In *Communications, Circuits and Systems, 2005. Proceedings. 2005 International Conference on*, volume 1, pages 71–73. IEEE, 2005.

[5] Paul Barrett. Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor. In *Advances in cryptology—CRYPTO'86*, pages 311–323. Springer, 1987.

[6] Chakib Bekara and Maryline Laurent-Maknavicius. A new resilient key management protocol for wireless sensor networks. In *Information Security Theory and Practices. Smart Cards, Mobile and Ubiquitous Computing Systems*, pages 14–26. Springer, 2007.

[7] Luca Benini, Giuliano Castelli, Alberto Macii, Enrico Macii, Massimo Poncino, and Riccardo Scarsi. A discrete-time battery model for high-level power estimation. In *Proceedings of the conference on Design, automation and test in Europe*, pages 35–41. ACM, 2000.

[8] Ernest F Brickell, Daniel M Gordon, Kevin S McCurley, and David B Wilson. Fast exponentiation with precomputation. In *Advances in Cryptology—EUROCRYPT'92*, pages 200–207. Springer, 1993.

[9] Daniel R. L. Brown. Sec 1 : Elliptic curve cryptography, May 2009.

[10] Michael Brown, Darrel Hankerson, Julio López, and Alfred Menezes. *Software implementation of the NIST elliptic curves over prime fields*. Springer, 2001.

[11] T Ryan Burchfield, S Venkatesan, and Douglas Weiner. Maximizing throughput in zigbee wireless networks through analysis, simulations and implementations. In *Proc. Int. Workshop Localized Algor. Protocols WSNs*, pages 15–29. Citeseer, 2007.

[12] Pamba Capochichi. *Conception d'une architecture hiérarchique de réseau de capteurs pour le stockage et la compression de données*. PhD thesis, UFR des Sciences et Techniques de l'Université de Franche-Comté, March 2010.

[13] Mihaela Cardei, Shuhui Yang, and Jie Wu. Algorithms for fault-tolerant topology in heterogeneous wireless sensor networks. *Parallel and Distributed Systems, IEEE Transactions on*, 19(4) :545–558, 2008.

[14] Certicom. Certicom intellectual property. https://www.certicom.com/index.php/\licensing/certicom-ip.

[15] Jinran Chen, Shubha Kher, and Arun Somani. Distributed fault detection of wireless sensor networks. In *Proceedings of the 2006 workshop on Dependability issues in wireless ad hoc networks and sensor networks*, pages 65–72. ACM, 2006.

[16] Chee-Yee Chong and Srikanta P Kumar. Sensor networks : evolution, opportunities, and challenges. *Proceedings of the IEEE*, 91(8) :1247–1256, 2003.

[17] Garth V Crosby and Niki Pissinou. Cluster-based reputation and trust for wireless sensor networks. In *Consumer Communications and Networking Conference*, 2007.

[18] Crossbow. Telosb mote platform, 2005.

[19] Crossbow. Micaz : Wireless measurement system, 2006.

[20] William M Daley and Raymond G Kammer. Digital signature standard (dss). Technical report, DTIC Document, 2000.

[21] Christophe De Cannière. Trivium : A stream cipher construction inspired by block cipher design principles. In *Information Security*, pages 171–186. Springer, 2006.

[22] Whitfield Diffie and Martin Hellman. New directions in cryptography. *Information Theory, IEEE Transactions on*, 22(6) :644–654, 1976.

[23] Min Ding, Dechang Chen, Kai Xing, and Xiuzhen Cheng. Localized fault-tolerant event boundary detection in sensor networks. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 2, pages 902–913. IEEE, 2005.

[24] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology*, pages 10–18. Springer, 1985.

[25] Eleazar Eskin, Andrew Arnold, Michael Prerau, Leonid Portnoy, and Sal Stolfo. A geometric framework for unsupervised anomaly detection. In *Applications of data mining in computer security*, pages 77–101. Springer, 2002.

[26] Youssou Faye, Herve Guyennet, Ibrahima Niang, and Yanbo Shou. Fast scalar multiplication on elliptic curve cryptography in selected intervals suitable for wireless sensor networks. In *Cyberspace Safety and Security*, pages 171–182. Springer, 2013.

[27] Michael Flynn. Some computer organizations and their effectiveness. *Computers, IEEE Transactions on*, 100(9) :948–960, 1972.

[28] Julien Franq. *Conception et sécurisation d'unités arthmétiques hautes performances pour courbes elliptiques*. PhD thesis, Université des Sciences et Techniques du Languedoc, Décembre 2009.

[29] David Gay, Philip Levis, Robert Von Behren, Matt Welsh, Eric Brewer, and David Culler. The nesc language : A holistic approach to networked embedded systems. In *Acm Sigplan Notices*, volume 38, pages 1–11. ACM, 2003.

[30] Jorge Guajardo and Christof Paar. *Efficient algorithms for elliptic curve cryptosystems*. Springer, 1997.

[31] Gaurav Gupta and Mohamed Younis. Fault-tolerant clustering of wireless sensor networks. In *Wireless Communications and Networking, 2003. WCNC 2003. 2003 IEEE*, volume 3, pages 1579–1584. IEEE, 2003.

[32] Nils Gura, Arun Patel, Arvinderpal Wander, Hans Eberle, and SheuelingChang Shantz. Comparing elliptic curve cryptography and rsa on 8-bit cpus. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 119–132. Springer Berlin Heidelberg, 2004.

[33] Dae-Man Han and Jae-Hyun Lim. Smart home energy management system using ieee 802.15. 4 and zigbee. *Consumer Electronics, IEEE Transactions on*, 56(3) : 1403–1410, 2010.

[34] D.R. Hankerson, S.A. Vanstone, and A.J. Menezes. *Guide to elliptic curve cryptography*. Springer-Verlag New York Inc, 2004.

[35] Wendi B Heinzelman, Anantha P Chandrakasan, and Hari Balakrishnan. An application-specific protocol architecture for wireless microsensor networks. *Wireless Communications, IEEE Transactions on*, 1(4) :660–670, 2002.

[36] Wendi Rabiner Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *System Sciences, 2000. Proceedings of the 33rd Annual Hawaii International Conference on*, pages 10–pp. IEEE, 2000.

[37] Jonathan W Hui and David Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 81–94. ACM, 2004.

[38] IEEE. Ieee standards 802.15.4 - part 15.4 : Wireless medium access control (mac) and physical layer (phy) specifications for low-rate wireless personal area networks (lr-wpans), October 2003.

[39] Nicholas Jansma and Brandon Arrendondo. Performance comparison of elliptic curve and rsa digital signatures, April 2004.

[40] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International Journal of Information Security*, 1(1) :36–63, 2001.

[41] P Johnson and DC Andrews. Remote continuous physiological monitoring in the home. *Journal of telemedicine and telecare*, 2(2) :107–113, 1996.

[42] Chris Karlof and David Wagner. Secure routing in wireless sensor networks : Attacks and countermeasures. *Ad hoc networks*, 1(2) :293–315, 2003.

[43] Priit Karu and Jonne Loikkanen. Practical comparison of fast public-key cryptosystems. In *Telecommunications Software and Multimedia Lab. at Helsinki Univ. of Technology, Seminar on Network Security*. Citeseer, 2001.

[44] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of computation*, 48(177) : 203–209, 1987.

[45] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in Cryptology—CRYPTO'99*, pages 388–397. Springer, 1999.

[46] Farinaz Koushanfar, Miodrag Potkonjak, and A Sangiovanni-Vincentell. Fault tolerance techniques for wireless ad hoc sensor networks. In *Sensors, 2002. Proceedings of IEEE*, volume 2, pages 1491–1496. IEEE, 2002.

[47] Farinaz Koushanfar, Miodrag Potkonjak, and Alberto Sangiovanni-vincentelli. Fault tolerance in wireless sensor networks. In *Handbook of Sensor Networks : Compact Wireless and Wired Sensing Systems*, 2004.

[48] Bhaskar Krishnamachari and Sitharama Iyengar. Distributed bayesian algorithms for fault-tolerant event region detection in wireless sensor networks. *Computers, IEEE Transactions on*, 53(3) :241–250, 2004.

[49] RSA Laboratories. Pkcs #1 : Rsa encryption standard, November 1993. URL ftp://ftp.rsasecurity.com/pub/pkcs/ascii/pkcs-1.asc.

[50] Koen Langendoen, Aline Baggio, and Otto Visser. Murphy loves potatoes. In *Proc. IPDPS*, 2006.

[51] Laurie Law, Alfred Menezes, Minghua Qu, Jerry Solinas, and Scott Vanstone. An efficient protocol for authenticated key agreement. *Designs, Codes and Cryptography*, 28(2) :119–134, 2003.

[52] Yee Wei Law, Jeroen Doumen, and Pieter Hartel. Survey and benchmark of block ciphers for wireless sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 2(1) :65–93, 2006.

[53] Jin-Shyan Lee, Yu-Wei Su, and Chung-Chou Shen. A comparative study of wireless protocols : Bluetooth, uwb, zigbee, and wi-fi. In *Industrial Electronics Society, 2007. IECON 2007. 33rd Annual Conference of the IEEE*, pages 46–51. IEEE, 2007.

[54] Arjen K Lenstra and Eric R Verheul. Selecting cryptographic key sizes. *Journal of cryptology*, 14(4) :255–293, 2001.

[55] Olivier Leran. Parallélisation d'un algorithme de chiffrement utilisant les courbes elliptiques. Master's thesis, Université de Franche-Comté, Décembre 2012.

[56] Philip Levis and David Culler. Maté : A tiny virtual machine for sensor networks. In *ACM Sigplan Notices*, volume 37, pages 85–95. ACM, 2002.

[57] Philip Levis, Sam Madden, Joseph Polastre, Robert Szewczyk, Kamin Whitehouse, Alec Woo, David Gay, Jason Hill, Matt Welsh, Eric Brewer, et al. Tinyos : An operating system for sensor networks. In *Ambient intelligence*, pages 115–148. Springer, 2005.

[58] Ning Li and Jennifer C Hou. Flss : a fault-tolerant topology control algorithm for wireless networks. In *Proceedings of the 10th annual international conference on Mobile computing and networking*, pages 275–286. ACM, 2004.

[59] Joshua Lifton, Mark Feldmeier, Yasuhiro Ono, Cameron Lewis, and Joseph A Paradiso. A platform for ubiquitous sensor deployment in occupational and domestic environments. In *Information Processing in Sensor Networks, 2007. IPSN 2007. 6th International Symposium on*, pages 119–127. IEEE, 2007.

[60] Chae Hoon Lim and Pil Joong Lee. More flexible exponentiation with precomputation. In *Advances in cryptology—CRYPTO'94*, pages 95–107. Springer, 1994.

[61] Kris Lin, Jennifer Yu, Jason Hsu, Sadaf Zahedi, David Lee, Jonathan Friedman, Aman Kansal, Vijay Raghunathan, and Mani Srivastava. Heliomote : enabling long-lived sensor networks through solar energy harvesting. In *Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 309–309. ACM, 2005.

[62] An Liu and Peng Ning. Tinyecc : A configurable library for elliptic curve cryptography in wireless sensor networks. In *Information Processing in Sensor Networks, 2008. IPSN'08. International Conference on*, pages 245–256. IEEE, 2008.

[63] Sergio Marti, Thomas J Giuli, Kevin Lai, and Mary Baker. Mitigating routing misbehavior in mobile ad hoc networks. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 255–265. ACM, 2000.

[64] K Martinez, P Padhy, A Riddoch, HLR Ong, and JK Hart. Glacial environment monitoring using sensor networks. In *In Proceedings of Real-World Wireless Sensor Networks*. Citeseer, 2005.

[65] David Martins. *Sécurité dans les réseaux de capteurs sans fil - Stéganographie et réseaux de confiance*. PhD thesis, UFR des Sciences et Techniques de l'Université de Franche-Comté, 2010.

[66] David Martins and Hervé Guyennet. Steganography in mac layers of 802.15. 4 protocol for securing wireless sensor networks. In *Multimedia Information Networking and Security (MINES), 2010 International Conference on*, pages 824–828. IEEE, 2010.

[67] James L Massey and Jimmy K Omura. Method and apparatus for maintaining the privacy of digital messages conveyed by public transmission, January 28 1986. US Patent 4,567,600.

[68] Marouane Mezzi. Réseaux de capteurs pou rla surveillance et la maintenance préventive d'une machine industrielle. Master's thesis, Université de Franche-Comté, Juin 2014.

[69] Victor S Miller. Use of elliptic curves in cryptography. In *Advances in Cryptology—CRYPTO'85 Proceedings*, pages 417–426. Springer, 1986.

[70] Sushruta Mishra, Lambodar Jena, and Aarti Pradhan. Fault tolerance in wireless sensor networks. *International Journal*, 2(10) :146–153, 2012.

[71] Atsuko Miyaji, Takatoshi Ono, and Henri Cohen. Efficient elliptic curve exponentiation. In *Proceedings of the First International Conference on Information and Communication Security*, pages 282–291. Springer-Verlag, 1997.

[72] Moteiv. Tmote sky : Ultra low power ieee 802.15.4 compliant wireless sensor module, 2005.

[73] James Newsome, Elaine Shi, Dawn Song, and Adrian Perrig. The sybil attack in sensor networks : analysis & defenses. In *Proceedings of the 3rd international symposium on Information processing in sensor networks*, pages 259–268. ACM, 2004.

[74] NIST. Recommended elliptic curves for federal government use, 1999. Available at http://csrc.nist.gov/encryption.

[75] Vladimir Oleshchuk and Vladimir Zadorozhny. Trust-aware query processing in data intensive sensor networks. In *Proc. of International Conference on Sensor Technologies and Applications (SensorComm 2007)*, 2007.

[76] Celal Ozturk, Yanyong Zhang, and Wade Trappe. Source-location privacy in energy-constrained sensor network routing. In *SASN*, volume 4, pages 88–93, 2004.

[77] Bibhudendu Panda and Pabitra Mohan Khilar. Fpga based implementation of parallel ecc processor. In *Proceedings of the 2011 International Conference on Communication, Computing & Security*, pages 453–456. ACM, 2011.

[78] Emile M Petriu, Nicolas D Georganas, Dorina C Petriu, Dimitrios Makrakis, and Voicu Z Groza. Sensor-based information appliances. *Instrumentation & Measurement Magazine, IEEE*, 3(4) :31–35, 2000.

[79] Joseph Polastre, Robert Szewczyk, and David Culler. Telos : enabling ultra-low power wireless research. In *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, pages 364–369. IEEE, 2005.

[80] Madhura Purnaprajna, Christoph Puttmann, and Mario Porrmann. Power aware reconfigurable multiprocessor for elliptic curve cryptography. In *Design, Automation and Test in Europe, 2008. DATE'08*, pages 1462–1467. IEEE, 2008.

[81] Kefa Rabah. Elliptic curve elgamal encryption and signature schemes. *Information Technology Journal*, 4(3) :299–306, 2005.

[82] Sutharshan Rajasegarar, Christopher Leckie, Marimuthu Palaniswami, and James C Bezdek. Distributed anomaly detection in wireless sensor networks. In *Communication systems, 2006. ICCS 2006. 10th IEEE Singapore International Conference on*, pages 1–5. IEEE, 2006.

[83] Daler Rakhamtov and Sarma Vrudhula. Time-to-failure estimation for batteries in portable electronic systems. In *Proceedings of the 2001 international symposium on Low power electronics and design*, pages 88–91. ACM, 2001.

[84] Brian Randell, Pete Lee, and Philip C. Treleaven. Reliability issues in computing system design. *ACM Computing Surveys (CSUR)*, 10(2) :123–165, 1978.

[85] Jussi Rautpalo. Gprs security-secure remote connections over gprs. *Technical University of Helsinki, Department of Computer Science, referenced*, 27 :2003, 2000.

[86] Kui Ren, Tieyan Li, Zhiguo Wan, Feng Bao, Robert H Deng, and Kwangjo Kim. Highly reliable trust establishment scheme in ad hoc networks. *Computer Networks*, 45(6) :687–699, 2004.

[87] Certicom Research. Sec 2 : Recommended elliptic curve domain parameters, September 2000.

[88] Vincent Ricquebourg, David Menga, David Durand, Bruno Marhic, Laurent Delahoche, and Christophe Loge. The smart home concept : our immediate future. In *E-Learning in Industrial Electronics, 2006 1ST IEEE International Conference on*, pages 23–28. IEEE, 2006.

[89] Ronald L Rivest, Adi Shamir, and Len Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2) : 120–126, 1978.

[90] Stanislav Rost and Hari Balakrishnan. Memento : A health monitoring system for wireless sensor networks. In *Sensor and Ad Hoc Communications and Networks, 2006. SECON'06. 2006 3rd Annual IEEE Communications Society on*, volume 2, pages 575–584. IEEE, 2006.

[91] Linnyer Beatrys Ruiz, Isabela G Siqueira, Hao Chi Wong, José Marcos S Nogueira, Antonio AF Loureiro, et al. Fault management in event-driven wireless sensor networks. In *Proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, pages 149–156. ACM, 2004.

[92] Stanislav Safaric and Kresimir Malaric. Zigbee wireless standard. In *Multimedia Signal Processing and Communications, 48th International Symposium ELMAR-2006, Zadar, Croatia*, pages 259–262, 2006.

[93] Yasuyuki Sakai and Kouichi Sakurai. Efficient scalar multiplications on elliptic curves without repeated doublings and their practical performance. In *Information Security and Privacy*, pages 59–73. Springer, 2000.

[94] Sergei P Skorobogatov and Ross J Anderson. Optical fault induction attacks. In *Cryptographic Hardware and Embedded Systems-CHES 2002*, pages 2–12. Springer, 2003.

[95] Jessica Staddon, Dirk Balfanz, and Glenn Durfee. Efficient tracing of failed nodes in sensor networks. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 122–130. ACM, 2002.

[96] Frank Stajano and Ross Anderson. The resurrecting duckling : security issues for ubiquitous computing. *Computer*, 35(4) :22–26, 2002.

[97] Robert Szewczyk, Alan Mainwaring, Joseph Polastre, John Anderson, and David Culler. An analysis of a large scale habitat monitoring application. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 214–226. ACM, 2004.

[98] Robert Szewczyk, Joseph Polastre, Alan Mainwaring, and David Culler. Lessons from a sensor network expedition. In *Wireless Sensor Networks*, pages 307–322. Springer, 2004.

[99] Ali Maleki Tabar, Arezou Keshavarz, and Hamid Aghajan. Smart home care network using sensor fusion and distributed vision-based reasoning. In *Proceedings of the 4th ACM international workshop on Video surveillance and sensor networks*, pages 145–154. ACM, 2006.

[100] Jane Tateson, Christopher Roadknight, Antonio Gonzalez, Taimur Khan, Steve Fitz, Ian Henning, Nathan Boyd, Chris Vincent, and Ian Marshall. Real world issues in deploying a wireless sensor network for oceanography. In *Workshop on Real-World Wireless Sensor Networks REALWSN'05*, 2005.

[101] John Thelen, Dann Goense, and Koen Langendoen. Radio wave propagation in potato fields. In *1st Workshop on Wireless Network Measurements*, volume 2, pages 331–338. Citeseer, 2005.

[102] Ben L Titzer, Daniel K Lee, and Jens Palsberg. Avrora : Scalable sensor network simulation with precise timing. In *Proceedings of the 4th international symposium on Information processing in sensor networks*, page 67. IEEE press, 2005.

[103] Gilman Tolle, Joseph Polastre, Robert Szewczyk, David Culler, Neil Turner, Kevin Tu, Stephen Burgess, Todd Dawson, Phil Buonadonna, David Gay, et al. A macroscope in the redwoods. In *Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 51–63. ACM, 2005.

[104] Wim Van Eck. Electromagnetic radiation from video display units : an eavesdropping risk ? *Computers & Security*, 4(4) :269–286, 1985.

[105] Maarten Van Steen. Distributed systems principles and paradigms. *Network*, 2 :28, 2004.

[106] Thiemo Voigt, Hartmut Ritter, and Jochen Schiller. Utilizing solar power in wireless sensor networks. In *Local Computer Networks, 2003. LCN'03. Proceedings. 28th Annual IEEE International Conference on*, pages 416–422. IEEE, 2003.

[107] Haodong Wang and Qun Li. Efficient implementation of public key cryptosystems on mote sensors (short paper). In *Information and Communications Security*, pages 519–528. Springer, 2006.

[108] Xun Wang, Sriram Chellappan, Wenjun Gu, Wei Yu, and Dong Xuan. Search-based physical attacks in sensor networks. In *Computer Communications and Networks, 2005. ICCCN 2005. Proceedings. 14th International Conference on*, pages 489–496. IEEE, 2005.

[109] Anthony Wood and John A Stankovic. Denial of service in sensor networks. *Computer*, 35(10) :54–62, 2002.

[110] Peter Wright and Paul Greengrass. *Spycatcher : The candid autobiography of a senior intelligence officer*. Viking New York, 1987.

[111] Zheng Yan, Peng Zhang, and Teemupekka Virtanen. Trust evaluation based security solution in ad hoc networks. In *Proceedings of the Seventh Nordic Workshop on Secure IT Systems*, volume 14, 2003.

[112] Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal. Wireless sensor network survey. *Computer networks*, 52(12) :2292–2330, 2008.

[113] Huafei Zhu, Feng Bao, and Robert H Deng. Computing of trust in wireless networks. In *Vehicular Technology Conference, 2004. VTC2004-Fall. 2004 IEEE 60th*, volume 4, pages 2621–2624. IEEE, 2004.

APPLICATION D'ECC DANS UN SYSTÈME DE SURVEILLANCE ENVIRONNEMENTALE

Sommaire

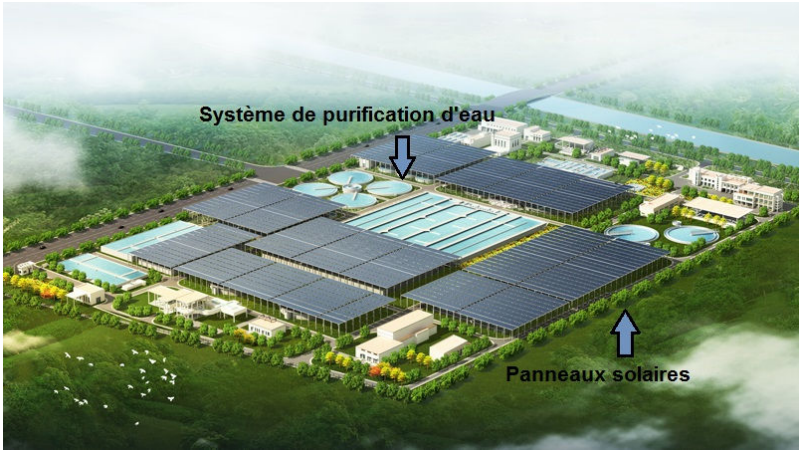


FIGURE 1 – Un système de purification d'eau alimenté par des panneaux solaires

Dans ce chapitre, nous présentons un projet d'un système de purification d'eau développé en Chine (voir figure 1), dont le fonctionnement peut être surveillé à distance avec un ensemble de capteurs. Pour sécuriser la communication entre les capteurs et la centrale de contrôle, on a choisi le cryptosystème asymétrique ECC, qui peut offrir une performance plus intéressante sur les systèmes embarqués. Afin d'accélérer le calcul, on a intégré notre solution de parallélisation des multiplications scalaires.

.1/ PRÉSENTATION DU PROJET

.1.1/ ARCHITECTURE DU SYSTÈME

L'objectif du projet de développement d'un système de purification d'eau qui peut fonctionner d'une manière autonome, et qui peut être contrôlé à distance. Les capteurs sont mis en place pour surveiller l'état de fonctionnement, et une unité de communication est aussi indispensable pour envoyer des données mesurées et recevoir des requêtes des utilisateurs. L'architecture du système est illustrée dans la figure 2.

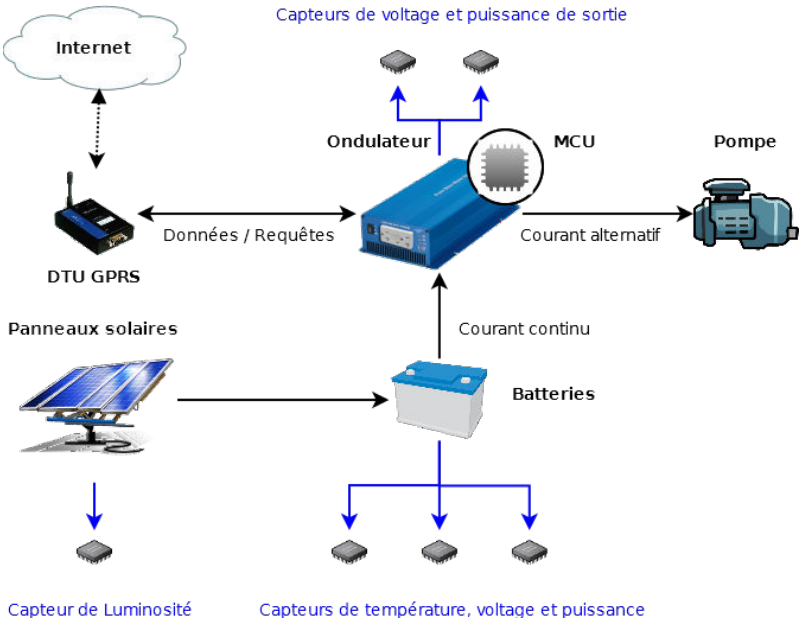


FIGURE 2 – Architecture de notre système de purification d'eau

Les panneaux solaires génèrent le courant continu et rechargent les batteries, un ondulateur connecté aux batteries convertit le courant continu au courant alternatif de 220 volts pour alimenter la pompe qui fait circuler l'eau.

Le projet peut être considéré comme un prototype et une expérimentation de l'Internet des objets. Pour automatiser et faciliter la gestion du système, un micro-contrôleur et un ensemble de capteurs sont mis en place pour surveiller les paramètres qui sont cruciaux pour son bon fonctionnement :

– Capteur de luminosité : le capteur est installé sur un panneau solaire pour mesurer le niveau de luminosité. Les panneaux convertissent le rayonnement solaire en énergie électrique, il n'y a donc aucun intérêt de les laisser allumés quand le niveau de luminosité est très faible.
– Capteur de température : Les batteries ne peuvent fonctionner que dans un intervalle de température particulier. Notamment une température trop élevée peut provoquer une incendie, même une explosion, il est donc indispensable de surveiller constamment la température des batteries.
– Capteur de voltage et puissance : le voltage et la puissance sont aussi 2 paramètres qui peuvent refléter l'état de fonctionnement du système. Un voltage ou une puissance très faible en entrée signifie probablement une panne de batterie, et un voltage ou une

puissance de sortie instable peut éventuellement endommager la pompe. Si on détecte des valeurs anormales, il faut arrêter immédiatement tout le système et envoyer un avertissement à la centrale de contrôle.

Les capteurs sont reliés au micro-contrôleurs (MCU) qui est installé dans l'ondulateur, et il est programmé pour récupérer les données mesurées et contrôler les autres composants du systèmes. Pour envoyer des données et recevoir des requêtes, le micro-contrôleur est connecté à une unité de transfert de données (DTU), qui peut se connecter à la centrale de contrôle à distance via le réseau GPRS.

Grâce à la connexion entre la centrale et le DTU, l'utilisateur peut programmer à distance les horaires de fonctionnement des panneaux solaires, éteindre ou allumer des panneaux et interroger un panneau spécifique pour récupérer les paramètres surveillés à n'importe quel moment.

.1.2/ PROCESSEUR EMBARQUÉ

Le micro-contrôleur utilisé est un modèle amélioré du micro-contrôleur MSC-51, qui est un modèle initialement développé par Intel dans les années 80. Il est réputé pour sa fiabilité et sa simplicité d'utilisation, et les développeurs peuvent programmer directement en C standard. De ce fait, les autres modèles compatibles et améliorés continuent à sortir, sont encore largement utilisés dans l'industrie.

Une comparaison des caractéristiques techniques principales avec le micro-contrôleur TI MSP430 utilisé par Telosb est donnée dans le tableau 1.

	MSC-51	MSP430
Fréquence :	12 MHz	8 MHz
RAM :	512 Octets	10 Ko
ROM :	16 Ko	48 Ko

TABLE 1 – Comparaison des caractéristiques techniques entre MSC-51 et MSP430

Nous pouvons constater que MCS-51 peut fonctionner un peu plus vite que MSP430, mais il possède beaucoup moins de RAM et de ROM. La ROM est MCS-51 peut être *étendue jusqu'à 64 Ko*, mais la consommation de mémoire devient tout de même un facteur critique lors du développement pour ce modèle de micro-contrôleur.

.1.3/ RÉSEAU DE CAPTEURS

Dans un site de purification d'eau, on installe souvent plusieurs panneaux solaires, et chacun est surveillé par un groupe de capteurs qui sont directement contrôlés par le processeur (micro-contrôleur) installé dans l'ondulateur. D'ailleurs, pour pouvoir envoyer et recevoir des données, il faut avoir au moins un DTU connecté. Le DTU est équipé d'une carte SIM qui lui permet d'accéder aux réseaux de l'opérateur de téléphonie mobile, et la transmission de données génère sans doute des frais supplémentaires. Pour une raison de coût de maintenance, on n'installe qu'un seul DTU pour un groupe de processeurs. Les processeurs sont reliés ensemble, et les capteurs constituent un véritable réseau de capteurs (voir figure 3).

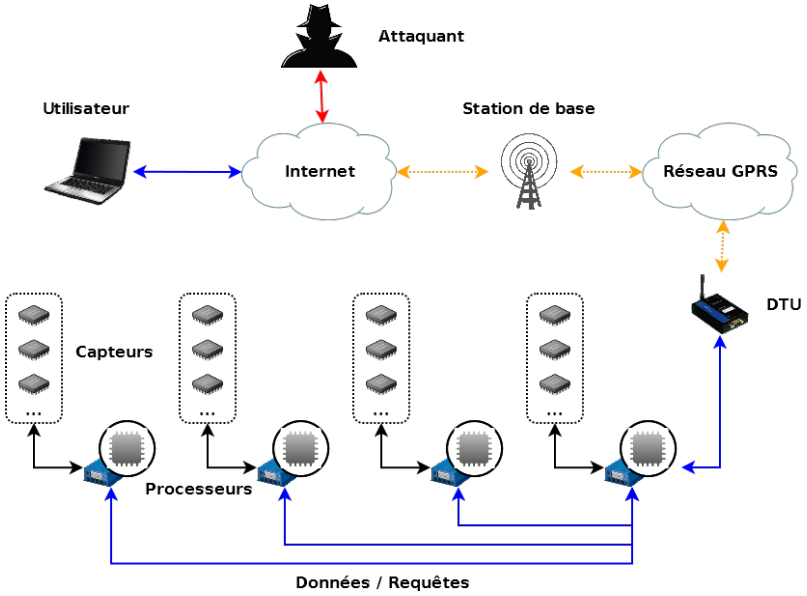


FIGURE 3 – Réseau de capteurs de notre système de purification d'eau

Les capteurs mesurent périodiquement des paramètres physiques, qui sont ensuite envoyés au processeur correspondant. Le processeur vérifie s'il a un DTU connecté, si oui, il lui envoie directement les données, sinon, les données sont d'abord envoyées au processeur possédant la connexion avec DTU avant d'être renvoyées au DTU.

Le DTU est un dispositif paramétrable, dans lequel on peut stocker les informations nécessaires pour établir une connexion avec la centrale, comme l'adresse IP du serveur, numéro de port, mode de connexion (TCP/UDP) etc. Les données sont envoyées à la station de base via le réseau GPRS, et puis arrivées à la centrale en passant par Internet.

Dans l'autre sens, l'utilisateur peut éteindre ou rallumer un composant spécifique en envoyant des requêtes aux processeurs à distance. Lorsque le DTU reçoit une requête, il l'envoie au processeur connecté, qui s'occupe de la redistribution de requêtes, s'il s'agit d'un autre processeur sur le réseau.

.2/ PROBLÈMES DE SÉCURITÉ DANS NOTRE SYSTÈME

Dans notre système, l'envoi de données et de requêtes est réalisé via une connexion sans fil entre le DTU et la centrale de contrôle. Les données doivent passer par 2 réseaux différents : Internet et le réseau GPRS qui est fourni par l'opérateur de téléphonie mobile.

La communication dans le réseau GPRS est sécurisée avec les algorithmes SRES [1] et GEA [2] [85], qui peuvent offrir un niveau de sécurité assez élevé. Cependant, une fois les données entrent sur Internet, il n'existe aucun mécanisme de sécurité inhérent pour les protéger contre l'attaquant de l'homme du milieu (voir figure 3).

L'objectif du projet est de développer un système qui peut fonctionner d'une manière au-

1. Signed Response for Subscriber Authentication
2. GPRS Encryption Algorithm

tonome, car ce genre de systèmes sont souvent installé en banlieue où une intervention humaine n'est pas toujours très facile. Il est donc indispensable de mettre en place des mécanismes de sécurité pour protéger les équipements et l'échange de données.

Le DTU lui-même ne détient aucun cryptosystème par défaut, si les données ne sont pas correctement chiffrées, l'attaquant pourra intercepter facilement les données qui circulent entre la centrale et le DTU, et lancer des attaques de réplication de données et de vol d'identité.

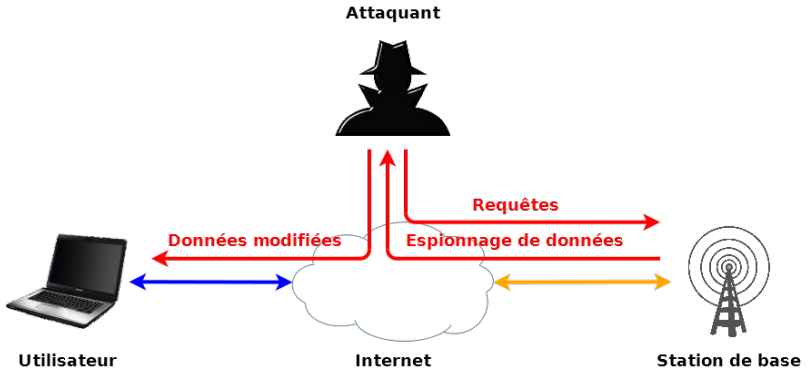


FIGURE 4 – Attaques dans notre système de purification d'eau

Pendant l'envoi de données mesurées, comme la communication n'est pas sécurisée, l'attaquant peut accéder à toutes les données qui circulent entre les 2 parties (voir figure 4).

Comme les données contiennent des informations qui représentent l'état de fonctionnement des équipements à distance, l'attaquant peut les modifier et renvoyer à l'utilisateur pour déclencher une fausse alarme. L'attaquant peut aussi lancer des attaquants de réplication de données. Il peut saboter les équipements en rediffusant les données qu'il a intercepté auparavant pour faire croire à l'utilisateur que tout se passe bien à distance.

Les attaques pendant l'envoie de requêtes peuvent aussi représenter une menace sérieuse au système. Chaque commande est représentée par une suite d'octets prédéfinie, en faisant des analyses, l'attaquant peut deviner les usages des commandes envoyées. Même les requêtes sont chiffrées, l'attaquant peut les intercepter et réutiliser sans chercher à déchiffrer les messages. Par exemple, l'attaquant peut répéter des requêtes pour éteindre et rallumer consécutivement les panneaux à distance pour les endommager.

Par conséquent, comme tous les réseaux, notre réseau de capteur est aussi vulnérable aux différentes attaques, il faut donc concevoir et appliquer des mécanismes de sécurité pour le protéger.

.3/ APPLICATION D'UN CRYPTOSYSTÈME BASÉ SUR ECC

Afin de protéger la communication entre la centrale de contrôle et le DTU, nous avons proposé d'implémenter un cryptosystème qui est basé sur l'ECC [69, 44], qui offre un ensemble de protocoles cryptographiques sophistiqués (voir section 3.4). Comme ceux d'Elgamal [24] qui nous permettent de chiffrer et de signer un message.

Cependant, pour utiliser les protocoles d'Elgamal, il faut d'abord trouver un algorithme

qui peut convertir le message à chiffrer à un point sur la courbe elliptique et vice versa. C'est faisable et il existe aussi des algorithmes dans la littérature, mais l'utilisation d'un tel algorithme implique aussi beaucoup plus de calculs mathématiques. Pour une raison de performance, dans la plupart des cas, on utilise plutôt un mécanisme hybride, dans lequel on utilise un algorithme de cryptographie asymétrique pour générer une clé secrète partagée, comme celui de Diffie-Hellman [22], ensuite on utilise un algorithme de cryptographie symétrique pour chiffrer le message. C'est aussi la stratégie choisie par le protocole ECIES (voir section 3.4.2).

Pour notre projet, nous voulons que dans un premier temps le cryptosystème puisse atteindre les objectifs suivants :

– Confidentialité : Les données qui circulent entre la centrale et le DTU sont correctement chiffrées et ne sont pas accessibles à l'attaquant ;
– Fraîcheur : Les données sont chiffrées avec un nonce et un horodatage (timestamp en Anglais) pour contre les attaques de réplication de données ;
– Authentification : Il faut que le récepteur du message puisse vérifier l'identité de l'envoyeur.

Dans notre système, les données circulent directement entre la centrale et le DTU sans passer par une tierce partie, afin d'authentifier les messages reçus, au lieu d'implémenter le protocole de signature numérique ECDSA (voir section 3.4.3) qui est sans doute une procédure trop coûteuse pour les micro-contrôleurs, on a décidé d'utiliser des codes d'authentification de message (MAC) qui peuvent être générés à partir de la clé secrète partagée.

.3.1/ CONFIGURATION DU CRYPTOSYSTÈME

On suppose que la centrale de contrôle et le micro-contrôleur qui est connecté au DTU possèdent chacun une clé privée et une clé publique (k_c, K_c) et (k_d, K_d) où k_c, k_d sont 2 nombres entiers de longueur 160. Les clés du micro-contrôleur sont chargées avant le déploiement, et celles de la centrale sont stockées sur un serveur qui peuvent être changées au cours du fonctionnement.

La centrale et les micro-contrôleurs partagent la même courbe elliptique définie sur un corps premier fini $\mathbb{F}_p$, les paramètres de la courbes à configurer sont donnés dans le tableau 6.1.

Pour accélérer le calcul des multiplications scalaires sur les micro-contrôleurs, on veut appliquer aussi notre solution de parallélisation, selon le test de performance (voir tableau 4.9), afin de paralléliser le calcul entre jusqu'à 4 processeurs, il faut précalculer et stocker au moins 3 points.

La consommation de mémoire pour stocker les paramètres est montrée dans le tableau 6.2, plus les clés (k_d, K_d), la quantité totale de mémoire nécessaire est 333 octet. Nous pouvons constater que le nombre total d'octets ne dépasse pas la taille maximale de ROM du micro-contrôleur MSC-51.

.3.2/ PROTOCOLE CRYPTOGRAPHIQUE

Chaque fois quand le DTU se connecte à la centrale de contrôle, il demande et télécharge la clé publique K_c de la centrale. Si la centrale veut changer ses clés, il suffira de couper la connexion actuelle, et le DTU va recréer la connexion et redemander la nouvelle clé publique.

Quand le micro-contrôleur veut chiffrer et envoyer un message m à la centrale, il effectue les opérations suivantes :

1. Générer un nombre entier aléatoire $r \in [1, n-1]$ et calculer $R = r \cdot G$;
2. Calculer le secret partagé $P = (P_x, P_y) = r \cdot K_c$;
3. Générer la clé partagée $s = KDF(P_x)$;
4. Chiffrer le message $c = E(s, (m, o, t))$ où o est le nonce et t est l'horodatage ;
5. Générer le code d'authentification de message $a = MAC(s, c)$;
6. Envoyer (R, c, a) à la centrale ;

$KDF()$ est une fonction de génération de clé, $MAC()$ une fonction de génération du code d'authentification, et $E()$ est un algorithme de cryptographie symétrique.

Quand la centrale reçoit le message (R, c, a), il essaie de le déchiffrer et authentifier en effectuant la procédure suivante :

1. Calculer le secret partagé $P = (P_x, P_y) = k_c \cdot R = k_c \cdot r \cdot G = r \cdot K_c$;
2. Générer la clé partagée $s = KDF(P_x)$;
3. Générer le code d'authentification de message $a' = MAC(s, c)$;
4. Déchiffrer le message $(m, o, t) = E^{-1}(s, c)$;
5. Comparer o et t avec le nonce n' et l'horodatage t' locaux ;
6. Rejeter le message si $n \neq n' + 1$ ou $t \leq t'$;
7. Rejeter le message si $a \neq a'$;

Quand la centrale veut envoyer une requête au micro-contrôleur à distance, il demande d'abord sa clé publique K_d, et la suite de la procédure est identique que celle présentée ci-dessus.

On suppose que l'attaquant possède l'accès à l'ensemble de paramètres de la courbe utilisée (p, a, b, G, n), et il peut aussi capturer le message (R, c, a) envoyé par le DTU. Comme il est extrêmement difficile de calculer r à partir de R, il ne peut pas calculer le clé partagée s pour déchiffrer le message c. Il ne peut pas modifier le message chiffré c non plus, car le code d'authentification a est généré avec la clé partagée s et le message chiffré c, un c modifié ne peut pas passer le test d'authentification.

D'ailleurs, l'attaquant ne peut réutiliser aucune des valeurs capturées. Les calculs de R et de a sont basés sur la même valeur r, qui n'est pas accessible à l'attaquant, et le message lui-même est protégé par un nonce o et un horodatage t, une réutilisation de message capturé sera détectée par le système.

.3.3/ CALCUL PARALLÈLE DES MULTIPLICATIONS SCALAIRES

Durant le chiffrement et le déchiffrement, chaque participant a une multiplication scalaire à effectuer, dans le chapitre 4, nous avons proposé une technique qui nous permet de paralléliser le calcul des multiplication scalaires en découpant le scalaire en plusieurs parties. Nous avons choisi la parallélisation avec des points précalculés dont la procédure est illustrée dans la figure 4.9, car la préparation des points en temps réel nécessite des calculs compliqués.

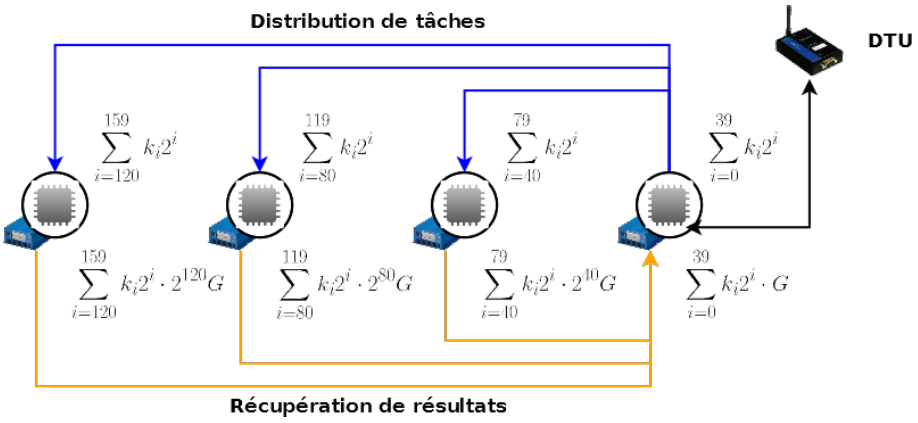


FIGURE 5 – Application de la parallélisation des multiplications scalaires

Le micro-contrôleur qui est connecté au DTU est considéré comme le processeur maître, et quand il veut effectuer une multiplication scalaire parallèle, il exécute des opérations suivantes :

1. Il envoie une demande de participation à tous les micro-contrôleurs connectés ;
2. Les micro-contrôleurs qui sont disponibles deviennent des esclaves en lui retournant des réponses positives ;
3. En fonction du nombre d'esclaves n, le maître génère un ensemble de tâches $t_i, i \in [0, n-1]$ en découpant le scalaire ;
4. Le maître envoie des tâches t_i aux esclaves, et le traitement des tâches commence ;
5. Après le traitement, les esclaves retournent les résultats $r_i, i \in [0, n-2]$ au maître ;
6. Le maître calcule le résultat final $R = \sum_{i=0}^{n} r_i$.

Par exemple, il y a 4 micro-contrôleurs qui sont sur le même réseau, et les points précalculés $2^{40}G, 2^{80}G, 2^{120}G$ sont déjà chargés en mémoire avant le déploiement. Quand le micro-contrôleur maître veut lancer le calcul parallèle, il découpe le scalaire en 4 parties $\sum_{i=120}^{159} k_i 2^i$, $\sum_{i=80}^{119} k_i 2^i$, $\sum_{i=40}^{79} k_i 2^i$ et $\sum_{i=0}^{39} k_i 2^i$, qui sont distribuées aux esclaves. Une fois le traitement des tâches est terminée, les résultats sont retournés au maître pour former le résultat final $\sum_{j=0}^{3} \sum_{i=40j}^{40j+40-1} k_i 2^i \cdot 2^{40j} G$ (voir figure 5).

.4/ CONCLUSION

Ce projet est une expérimentation de la parallélisation des multiplications scalaires au sein d'un véritable réseau de capteurs. La transmission de données et la manipulation

des panneaux solaires sont déjà opérationnelles, mais le développement de la partie cryptographique est toujours en cours, où nous étudions la possibilité d'implémenter le protocole de signature numérique ECDSA.

DÉROULEMENT DE L'ALGORITHME D'EUCLIDE ÉTENDU

Dans cette annexe, nous montrons un exemple de calcul de l'inverse modulaire en utilisant l'algorithme d'Euclide étendu. Nous supposons que nous avons 2 nombres entiers 2013 et 313, nous appliquons d'abord l'algorithme d'Euclide pour calculer leur PGCD.

Dividende	-	Quotient	×	Diviseur	=	Reste
2013	–	6	×	313	=	135
313	–	2	×	135	=	43
135	–	3	×	43	=	6
43	–	7	×	6	=	1
6	–	6	×	1	=	0

TABLE 2 – Exemple de l'algorithme d'Euclide

Nous prenons le dernier reste avant le zéro, le $PGCD(2013, 313) = 1$, c'est-à-dire que 2013 et 313 sont premier entre eux. Nous pouvons voir que à partir de la 2e ligne, le diviseur est le reste de la ligne précédente, et à partir de la 3e ligne, le dividende est égal au reste obtenu 2 ligne plus haut. Nous pouvons donc réécrite le tableau 2 de la manière suivante.

$u \times x + v \times p =$			r
$1 \times 2013 + 0 \times 313 =$			2013
$0 \times 2013 + 1 \times 313 =$			313
$1 \times 2013 - 6 \times 313 =$		$2013 - 6 \times 313 =$	135
$-2 \times 2013 + 7 \times 313 =$	$313 - 2 \times 135 =$	$313 - 2 \times (2013 - 6 \times 313) =$	43
$7 \times 2013 - 27 \times 313 =$	$135 - 3 \times 43 =$	$(2013 - 6 \times 313) - 3 \times (-2 \times 2013 + 7 \times 313) =$	6
$-51 \times 2013 + 196 \times 313 =$	$43 - 7 \times 6 =$	$(-2 \times 2013 + 7 \times 313) - 7 \times (7 \times 2013 - 27 \times 313) =$	1

TABLE 3 – Exemple de l'algorithme d'Euclide étendu

Nous obtenons dans le tableau 3 les coefficients de Bézout $u \times x + v \times p = -51 \times 2013 + 196 \times 313 = 1$, nous avons donc l'inverse multiplicatif de 2013 modulo 313 est égal à -51, noté $-51 \equiv 2013^{-1} \pmod{313}$. Maintenant il nous suffit appliquer encore une fois l'opération modulo pour limiter le résultat dans l'intervalle $[0, 312]$, c'est-à-dire $262 \equiv -51 \pmod{313}$.

Résumé :

L'émergence des systèmes embarqués a permis le développement des réseaux de capteurs sans fil dans de nombreux domaines différents. Cependant, la sécurité reste un problème ouvert. La vulnérabilité des nœuds est principalement liée au manque de ressources. En effet, l'unité de traitement ne dispose pas d'assez de puissance et de mémoire pour gérer des mécanismes de sécurité très complexes.
La cryptographie est une solution qui est largement utilisée pour sécuriser les réseaux. Par rapport à la cryptographie symétrique, la cryptographie asymétrique nécessite des calculs plus compliqués, mais elle offre une distribution de clés plus sophistiquée et la signature numérique.
Dans cette thèse, nous essayons d'optimiser la performance d'ECC (Elliptic Curve Cryptography), un cryptosystème asymétrique qui est connu pour sa robustesse et son utilisation de clé plus courte par rapport à RSA. Nous proposons d'utiliser le parallélisme pour accélérer le calcul de la multiplication scalaire, qui est reconnue comme l'opération la plus coûteuse sur les courbes elliptiques. Les résultats de tests ont montré que notre solution offre un gain intéressant malgré une augmentation de la consommation d'énergie.
La deuxième partie de la contribution concerne l'application de la tolérance aux pannes dans notre architecture de parallélisation. Nous utilisons les nœuds redondants pour la détection des pannes et la restauration du calcul. Ainsi, en utilisant l'ECC et la tolérance aux pannes, nous proposons une solution de sécurité efficace et sûre pour les systèmes embarqués.

Mots-clés : Réseaux de capteurs sans fil, sécurité, cryptographie, courbe elliptique, parallélisme, tolérance aux pannes

Abstract:

The emergence of embedded systems has enabled the development of wireless sensor networks in different domains. However, the security remains an open problem. The vulnerability of sensor nodes is mainly due to the lack of resources. In fact, the processing unit doesn't have enough power or memory to handle complex security mechanisms.
Cryptography is a widely used solution to secure networks. Compared with symmetric cryptography, the asymmetric cryptography requires more complicated computations, but it offers more sophisticated key distribution schemes and digital signature.
In this thesis, we try to optimize the performance of ECC. An asymmetric cryptosystem which is known for its robustness and the use of shorter keys than RSA. We propose to use parallelism techniques to accelerate the computation of scalar multiplications, which is recognized as the most computationally expensive operation on elliptic curves. The test results have shown that our solution provides a significant gain despite an increase in energy consumption.
The 2nd part of our contribution is the application of fault tolerance in our parallelism architecture. We use redundant nodes for fault detection and computation recovery. Thus, by using ECC and fault tolerance, we propose an efficient and reliable security solution for embedded systems.

Keywords: Wireless sensor networks, security, cryptography, elliptic curve, parallelism, fault tolerance

SPIM

■ École doctorale SPIM 16 route de Gray F - 25030 Besançon cedex
■ tél. +33 (0)3 81 66 66 02 ■ ed-spim@univ-fcomte.fr ■ www.ed-spim.univ-fcomte.fr

UFC
UNIVERSITÉ DE FRANCHE-COMTÉ