Influence d'une architecture de type maître-esclave dans les problématiques de sécurité de l'Internet des objets

Philippe Pittoli

Université de Strasbourg

Thèse réalisée dans le Laboratoire ICube en vue de l'obtention du grade de Docteur en Informatique

---

Influence d'une architecture de type maître-esclave dans les problématiques de sécurité de l'Internet des Objets

---

Auteur :
Philippe Pittoli

Jury :
Thomas Noël : Directeur de thèse
Pierre David : Encadrant
Isabelle Chrisment : Rapportrice
Nathalie Mitton : Rapportrice
André-Luc Beylot : Examinateur
Franck Rousseau : Examinateur

21 mai 2019

Remerciements

Je tiens avant tout à remercier les personnes qui auront le courage de lire ce document. Parmi les personnes les plus méritantes, je souhaite adresser un remerciement des plus sincères à mon directeur de thèse, Thomas Noël, et mon co-encadrant de thèse Pierre David, pour avoir eu la patience de m'accompagner dans mon travail et en particulier d'avoir lu et relu patiemment mes travaux. Leurs conseils, leurs indications et leurs encouragements se sont avérés précieux pour le bon déroulement de cette thèse. Je tiens également à remercier Julie Nuguet, qui au-delà d'une relecture intense de ma thèse, m'a encouragé et réconforté durant les quatre longues années de cette aventure, et durant toutes mes années d'études supérieures.

J'adresse également un immense remerciement à mes amis pour ne pas m'avoir laissé tombé alors que nos rencontres s'espaçaient, et en particulier à Rémy Silber, Luka Vandervelden et Guillaume Lucas pour leurs encouragements, leur aide et leur patience.

Pour finir, je remercie les membres de mon jury de thèse pour l'intérêt qu'ils ont porté à mon travail et simplement pour avoir accepté d'être rapportrices, Mesdames Isabelle Chrisment et Nathalie Mitton, ou examinateurs, Messieurs André-Luc Beylot et Franck Rousseau.

Table des matières

Table des figures

Liste des tableaux

Liste des Sigles

6LoWPAN	IPv**6** over **Lo**w-Power **W**ireless **P**ersonal **A**rea **N**etworks
6LoWPAN-GHC	6LoWPAN **G**eneric **H**eader **C**ompression
AEAD	**A**uthenticated **E**ncryption with **A**dditional **D**ata
AES	**A**dvanced **E**ncryption **S**tandard
AH	**A**uthenticated **H**eader
ASN.1	**A**bstract Syntax **N**otation One
BER	**B**asic **E**ncoding **R**ules
CBC-MAC	Using CBC and MAC to encrypt and authenticate a message
CBC	**C**ipher **B**lock **C**haining
CBOR	**C**oncise **B**inary **O**bject **R**epresentation
CCM	**C**ounter **C**BC-**M**AC
CoAP	**Co**nstrained **A**pplication **P**rotocol
COSE	**C**BOR **O**bject **S**igning and **E**ncryption
CTS	**C**iphertext **S**tealing
DHE	**E**phemeral **D**iffie-**H**ellman
DH	**D**iffie-**H**ellman
(D)TLS	(**D**atagram) **T**ransport **L**ayer **S**ecurity, protocole de communication sécurisée
ECDHE	**E**phemeral **E**lliptic **C**urve **D**iffie-**H**ellman
ECDH	**E**lliptic **C**urve **D**iffie-**H**ellman
ESP	**E**ncapsulating **S**ecurity **P**ayload
ICV	**I**ntegrity **C**hecking **V**alue
id	identifiant ou identité
IEEE	**I**nstitute of **E**lectrical and **E**lectronics **E**ngineers, association de professionnels éditrice de journaux, publie également des tutoriels et des standards en rapport avec l'ingénieurie électrique
IETF	**I**nternet **E**ngineering **T**ask **F**orce, Organisme de standardisation
IKE	**I**nternet **K**ey **E**xchange
IPsec	Protocole de communication sécurisée
Kio	**K**ib**io**ctets
MAC	**M**essage **A**uthentication **C**ode ou **M**edia **A**ccess **C**ontrol
MQTT-SN	**MQTT** for **S**ensor **N**etworks
MQTT	**M**essage **Q**ueuing **T**elemetry **T**ransport
NAT	**N**etwork **A**ddress **T**ranslation
nonce	***N**umber used once*, une suite d'octets unique
OSCAR	**O**bject **S**ecurity **A**rchitecture for the **I**nternet of **T**hings
PDR	**P**acket **D**elivery **R**atio

PFS	**P**erfect **F**orward **S**ecrecy
PRF	**P**seudo **R**andom **F**unction, fonction permettant de générer une chaîne de taille arbitraire de contenu (presque) aléatoire à partir d'une graine.
Proxy	Serveur mandataire
SSH	**S**ecure **SH**ell

Chapitre 1

Introduction

Jusqu'à récemment, les réseaux étaient majoritairement composés d'ordinateurs. Depuis 1969, et l'avènement du premier réseau à transfert de paquets nommé « Advanced Research Projects Agency Network » (ARPAnet), le nombre de machines n'a cessé d'augmenter. En 2003, plus de 500 millions de machines étaient déployées sur Internet dans le monde [1]. L'avancée de la technologie a permis l'émergence de nouveaux périphériques comme les smartphones et les tablettes. La miniaturisation des composants et l'électronique bon marché ont également permis la création massive de tout petits ordinateurs pouvant communiquer, les personnes et les objets sont désormais connectés. Depuis 2008, nous avons plus d'appareils connectés que d'êtres humains sur terre [2], et nous parlons maintenant de réseaux d'ordinateurs et d'objets : l'Internet des Objets (IoT).

L'IoT inclut les ordinateurs (portables, fixes) mais aussi des appareils contraints (en énergie, puissance de calcul, ou mémoire). Certains de ces appareils peuvent récupérer des informations via des capteurs, puis communiquer entre eux, cela forme un réseau de capteurs. Ces réseaux ont des applications améliorant notre qualité de vie en automatisant des tâches à la maison, en surveillant la qualité de l'air, ou nous permettent des économies d'énergie en éteignant les lampadaires dans les rues sans piétons, ou encore nous permettent la détection de feux de forêt...

Ces réseaux sont parfois critiques, par exemple pour l'ouverture d'une porte d'entreprise, le lancement d'une alerte, ou encore les applications médicales. Cette criticité implique la sécurisation des communications et des appareils, une communication rapide notamment lorsqu'il y a une interaction humaine en direct (par exemple pour allumer ou éteindre une lampe), et un code simple pour éviter des erreurs.

1.1 L'Internet des Objets

Un des principes de base de l'IoT est que les objets sont accessibles sur Internet. Un client peut demander des informations à des objets, comme la température d'une pièce. Les objets fournissant des données sont des *capteurs*. Un client peut également envoyer une commande à des objets, comme l'ouverture d'une porte. Les objets permettant le contrôle physique d'un mécanisme sont des *actionneurs*.

Ces objets sont souvent contraints matériellement, ils ont des contraintes sur le logiciel, des contraintes sur le réseau et sur l'énergie disponible. Ces objets contraints peuvent être sur un réseau séparé du réseau de l'entreprise (ou de la maison, par exemple), et une passerelle est utilisée pour les relier à Internet.

Les domaines d'application possibles de ces objets sont multiples [3] :

— environnement : l'étude du climat [4], la détection de feux de forêt, la surveillance de la pluviométrie et de l'activité sismique ;

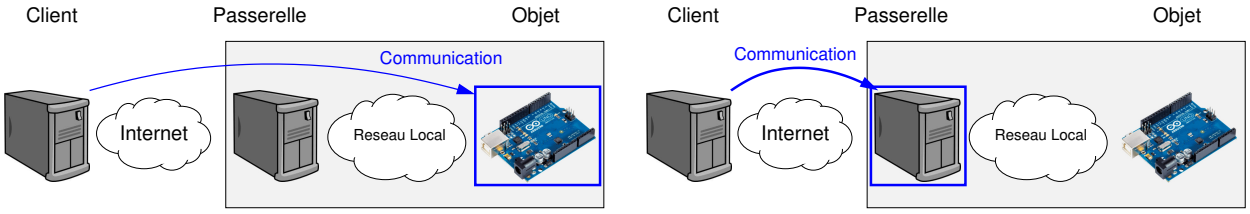
— santé : la surveillance de patients (pulsations cardiaques, tension, etc.) [5] ;

— transport : la surveillance, la localisation et le conditionnement de produits ;
— industrie : la surveillance des outils, le contrôle de la qualité des produits, etc. ;
— domotique : l'automatisation de la maison (ouvrir des volets, allumer et éteindre des lumières, etc.) ;
— urbain : la mesure de la pollution [6] ou de la qualité de l'eau [7], du trafic routier, la gestion intelligente des lumières, etc. ;
— militaire : la détection et la localisation de tireurs embusqués [8], la détection d'attaques, la reconnaissance de terrains ;
— réseau de distribution d'électricité (smart grid), pour mieux répartir la charge sur le réseau électrique.

Certains de ces déploiements sont critiques, par exemple dans les applications médicales. De plus, les objets connectés sont nombreux et peuvent être placés à des endroits peu accessibles physiquement (en zone de conflits, en hauteur dans un entrepôt, etc.). Leur maintenance doit par conséquent être réduite, par exemple en réduisant la complexité matérielle et logicielle des objets. Certains domaines d'application nécessitent de prendre en compte la sécurité, que ce soit pour la récupération de données ou l'envoi de commandes. Cette criticité implique de sécuriser les communications de bout en bout, entre le client et le réseau contraint, et entre les objets au sein du réseau.

1.2 Problématique de sécurité : la terminaison de la connexion

Les réseaux contraints sont composés de nœuds (des objets) et d'un routeur faisant la passerelle entre le réseau et Internet. Les clients veulent des informations fournies par les nœuds contraints, donc ils se connectent au réseau contraint, ce qui peut se faire de deux manières. La figure 1.1 illustre les deux méthodes de connexion au réseau.



(A) Architecture de référence.

(B) Architecture maître-esclave.

FIGURE 1.1 – Réseau IoT exemple. Si le client communique directement avec l'objet, c'est une architecture de référence. S'il communique avec la passerelle, c'est une architecture maître-esclave.

La première méthode, illustrée par la figure 1.1a, consiste pour le client à se connecter directement à l'objet. Le client est en contact avec l'objet, il lui demande des informations, et lui envoie des commandes. Nous retrouvons ce fonctionnement dans les réseaux d'ordinateurs, comme sur Internet : un client souhaite accéder à une page web, il se connecte à un ordinateur qui lui envoie la page demandée. Cette manière d'accéder à une ressource est utilisée au quotidien, et représente une *architecture de référence*. Une autre manière de procéder, illustrée par la figure 1.1b, consiste pour le client à se connecter à la passerelle, qui se charge de la communication avec les objets. L'objet n'est pas indépendant et il reçoit des ordres de la passerelle et uniquement de la passerelle, ce qui implique que le client ne communique pas directement avec l'objet mais demande à la passerelle qui ensuite envoie des ordres à l'objet. Cette architecture est dite *maître-esclave* : l'objet est un esclave de la passerelle (le maître). Ces deux méthodes impliquent des paradigmes de sécurité différents.

Dans une **architecture de référence**, la terminaison de la connexion se fait à l'objet, ce qui implique que :

1. l'objet connaît tous ses clients, ou les clients partagent des identifiants ;

2. l'objet utilise de la cryptographie pour authentifier les clients ;
3. l'objet est directement accessible, donc peut être vulnérable à cause de ses contraintes ou de celles du réseau.

Dans une **architecture maître-esclave**, la terminaison de la connexion se fait à la passerelle qui relie le réseau contraint à Internet, donc :

1. l'authentification se fait sur la passerelle, qui n'est pas contrainte ;
2. l'objet n'a pas à connaître les clients ;
3. l'objet n'a qu'une connexion sécurisée, avec la passerelle ;
4. l'objet n'est pas accessible directement, il ne répond qu'à la passerelle, qui elle reçoit les requêtes.

1.3 Motivation et hypothèses

Notre objectif est de comprendre l'influence d'une architecture de type maître-esclave dans les communications de l'IoT. Tout d'abord, nous souhaitons analyser la sécurité offerte par cette architecture. Ensuite, nous analysons les fonctionnalités permises et les limitations de cette architecture. Enfin, nous regardons son adéquation avec les contraintes des matériels et du réseau, ce qui implique la vérification de la mémoire utilisée, de la taille du code, des temps de calcul, mais également des temps de connexion et de communication (critiques dans certaines applications).

Deux hypothèses forment notre postulat de départ :

1. les algorithmes sont considérés parfaits : nous ne ferons pas de critique directe de la cryptographie, et toute donnée, chiffrée ou authentifiée avec un algorithme sans faille documentée et prouvée, est considérée protégée.
2. l'énergie n'est pas une contrainte : l'industrie 4.0 n'a pas de problème d'énergie, les nœuds contraints sont connectés à des machines outils et au secteur. Par conséquent, nous pouvons nous passer de cette contrainte [1]. Cependant, nous donnerons des pistes de réflexion quant à cette problématique.

1.4 Structure du document

Le chapitre 2 illustre les technologies employées dans les environnements contraints, c'est-à-dire les contraintes matérielles et les technologies de communication adaptées à ces contraintes (de la couche physique à la couche application). Ce chapitre présente également notre environnement de référence, sur lequel nous nous reposerons pour analyser et comparer les architectures. Les principes de base de la sécurité sont expliqués au chapitre 3. Ces principes couvrent la sécurité des communications, comme le chiffrement, l'authenticité, l'intégrité des messages par exemple, mais aussi la sécurité dans un sens plus général, comme l'authentification, l'autorisation, et les attaques possibles dans un réseau. Connaître ces attaques permet de mieux comprendre la portée de nos contributions. Une définition de ce qu'est une architecture de sécurité est donnée au chapitre 4, illustrant également les propriétés minimales d'une architecture sécurisée. Des architectures correspondant à cette définition sont listées dans le chapitre 5. Ce même chapitre introduit des scénarii, permettant de comprendre les différences entre ces architectures dans des cas d'usage courants. Enfin, ce chapitre compare ces architectures, selon de nombreux critères, allant de la sécurisation des communications aux contraintes imposées sur les nœuds. Suite à cette comparaison, le chapitre 6 fournit une étude de compatibilité des architectures avec des contraintes de déploiement, en apportant une conclusion à l'état de l'art des architectures de sécurité. Une optimisation d'une architecture de référence est présentée et évaluée au chapitre 7. Une architecture maître-esclave (CASAN) est présentée au chapitre 8 : pensée pour les environnements contraints, elle se compose d'un maître (nœud non contraint) et d'esclaves (les objets connectés). Elle

1. Bien que, de manière indirecte, la réduction du code implique bien souvent une réduction d'énergie consommée.

permet à la fois de récupérer des données de capteurs, mais aussi d'envoyer des commandes à des actionneurs, tout en prenant en compte la différence de capacités (de calcul et de mémoire) entre le maître et les objets connectés. Une amélioration des communications au sein du réseau contraint de l'architecture maître-esclave CASAN est présentée dans ce même chapitre 8. Une validation expérimentale de l'architecture maître-esclave ainsi améliorée est présentée au chapitre 9, prenant en compte des métriques liées aux contraintes des réseaux (temps de démarrage du réseau, durée d'un échange...). Enfin, le chapitre 10 présente un bilan de nos contributions et ouvre des perspectives pour des travaux futurs.

1.5 Contributions

Les objectifs de nos contributions sont de mettre en avant un état de l'art des architectures actuelles pour les environnements contraints, et de comparer les deux modèles d'architecture.

Notre première contribution est triple : définition des architectures de sécurité, état de l'art des architectures existantes et recommandations en fonction des contraintes de déploiement. La définition fournit un langage commun pour caractériser le fonctionnement d'une architecture, et nous aide à trouver les architectures de sécurité dans la littérature. Puis, dans un contexte d'environnement contraint basé sur des protocoles standards, et selon deux scénarii, nous comparons ces architectures. La comparaison est réalisée suivant un ensemble de critères liés aux contraintes imposées sur le matériel et le réseau, ainsi que sur la sécurité des architectures proposées. Cette comparaison forme un état de l'art de ces architectures. Suite à cet état de l'art, des recommandations sont émises selon des contextes de déploiement.

Notre seconde contribution apporte une optimisation d'une architecture de sécurité de référence. Dans ce type d'architecture, seuls un client et un serveur s'échangent des messages. Par conséquent, la durée de connexion est directement liée au protocole de communication sécurisé utilisé. Notre optimisation porte sur le protocole DTLS, un protocole standard et très répandu pour sécuriser les communications dans les réseaux contraints.

Notre troisième contribution porte sur l'optimisation d'une architecture maître-esclave et sa validation. Pour cela, nous partons de l'architecture CASAN, une architecture conçue dans le laboratoire ICube, qui utilise un maître non contraint pour recevoir les connexions des clients, et fait l'interface avec le réseau composé d'esclaves contraints, puis nous y apporterons une optimisation pour donner CASAN-S. Cette architecture est comparée à une architecture de référence, basée sur les protocoles CoAPS, 6LoWPAN et RPL, qui sont les standards dans les environnements contraints. La comparaison porte sur des critères de vitesse de connexion, de requête, ainsi que sur les contraintes des matériels et les fonctionnalités de ces architectures.

Chapitre 2

Environnements contraints

Notre travail est lié à la protection des communications, dans des environnements contraints, c'est-à-dire des réseaux composés d'objets ayant des capacités limitées. Dans ce chapitre, nous introduisons ces environnements, c'est-à-dire les contraintes associées aux objets, puis les protocoles standardisés prenant en compte ces contraintes. Parmi les technologies présentées, nous détaillons un ensemble cohérent de protocoles pour former l'environnement cible de nos travaux.

Ce chapitre présente un réseau IoT représentatif du contexte de notre étude :

1. Les contraintes matérielles sur les nœuds (10 Kio mémoire, 100 Kio Flash, 16 MHz) ;
2. La justification de IEEE 802.15.4 comme technologie radio, 6LoWPAN comme couche réseau et CoAP comme couche applicative ;
3. Une architecture représentative d'un déploiement.

2.1 Contraintes sur les objets

Nous détaillons dans un premier temps les contraintes matérielles associées aux objets que nous prenons pour cible dans ce document. Les objets utilisés en tant que capteurs et actionneurs sont limités en vitesse de calcul, mémoire et réseau. Les objets ont des micro-contrôleurs de quelques MHz à quelques dizaines de MHz, ce qui implique d'utiliser des algorithmes de faible complexité, s'exécutant rapidement.

	RAM	code
Classe 0	$\ll$ 10 Kio	$\ll$ 100 Kio
Classe 1	$\approx$ 10 Kio	$\approx$ 100 Kio
Classe 2	$\approx$ 50 Kio	$\approx$ 250 Kio

Table 2.1 – Classes d'objets contraints selon la RFC 7228.

Les objets sont catégorisés en classes par l'IETF [9], identifiant les objets par leurs contraintes de mémoire (RAM et code) comme nous le voyons dans le tableau 2.1. Ces contraintes impliquent de limiter les états à maintenir (stockés en RAM, de quelques Kio à quelques dizaines de Kio), et le nombre d'instructions du code (stockés en mémoire Flash, de quelques dizaines de Kio à environ 250 Kio).

Les objets peuvent être limités en énergie, ce qui implique de diminuer les calculs et réveiller les objets le moins possible (par exemple en simplifiant les protocoles pour échanger le moins de messages possible).

Enfin, les objets contraints ont également une capacité de communication limitée, et forment des réseaux également qualifiés de contraints. Ces réseaux ont souvent un débit, un nombre et une taille de message limités.

Nous nous concentrons sur des réseaux d'objets de classe 1, avec une fréquence d'horloge du micro-contrôleur autour de 16 MHz, sans matériel spécifique pour accélérer les calculs cryptographiques. Ces objets acceptent la cryptographie et sont bon marché, ils peuvent être largement déployés.

2.2 Couches réseau physique et liaison

Dans cette section, nous détaillons le choix de IEEE 802.15.4 comme couches physique et liaison. Ce choix est basé sur le fait qu'il s'agisse de couches standardisées, ouvertes et répandues. Le débit recherché est faible, pour limiter la consommation d'énergie. La portée nécessaire est de l'ordre de cent mètres, afin de pouvoir facilement déployer des nœuds par exemple dans un entrepôt ou une maison. Enfin, nous souhaitons étudier une architecture indépendante des couches physique et liaison : la technologie fournissant ces couches doit permettre d'utiliser des protocoles standards de plus haut niveau pour fournir les couches réseau, transport, application et la sécurité.

2.2.1 Introduction

La couche physique permet d'échanger des octets sur un médium (sur un fil, ou dans l'air). Cela se traduit par une carte réseau ou une puce radio, ce que nous retrouvons souvent dans les réseaux contraints. Dans ces réseaux, nous qualifions la radio employée comme « low power radio » [10]. Les problèmes liés à cette couche physique sont nombreux. Par exemple, la distance entre les nœuds et la qualité du lien ne sont pas corrélées : des nœuds proches peuvent avoir une mauvaise communication (un débit faible, voire une absence de connectivité), et inversement pour des nœuds éloignés. De plus, la qualité des liens est très dépendante de l'environnement (humidité, température, nombre de nœuds communiquant, présence humaine, obstacles, etc.). Enfin, l'usage de fréquences dans la bande des 2,4 GHz implique des interférences possibles avec d'autres réseaux, tels que 802.11 (WiFi).

La couche liaison apporte l'adressage des nœuds[1], et gère le médium partagé. Cette gestion implique de savoir qui peut accéder au médium et quand, de gérer les collisions et les interférences. La couche liaison est également responsable de l'énergie consommée par la radio, qui constitue une part non négligeable de l'énergie d'un objet contraint. Par exemple, si on sait à l'avance qui doit communiquer et à quel moment, les autres nœuds peuvent être éteints. Cette économie d'énergie est importante dans les réseaux avec des objets sur batteries ; certaines technologies annoncent une durée de vie des objets supérieure à 10 ans (comme NB-IoT, EC-GSM-IoT, LoRaWAN) sur une batterie modeste (équivalente à deux piles AAA).

2.2.2 État de l'art

La figure 2.1 liste des technologies existantes. Nous pouvons différencier trois grandes catégories, suivant la portée (courte, moyenne ou longue). Le débit est également une métrique utilisée pour faire ressortir les technologies orientées vers des usages privilégiant le débit à l'économie d'énergie.

Les technologies de longue portée (de 2 à 100 km) sont axées sur la récolte de données, et ont d'importantes contraintes concernant les communications. Ces contraintes concernent par exemple le nombre de paquets échangés (seulement quelques dizaines de paquets par jour par exemple), le débit, la vitesse d'échange ou encore la taille des paquets. Parmi ces technologies, nous pouvons citer :

- les technologies cellulaires : NB-IoT, EC-GSM-IoT, LTE-M ;
  Ces technologies ne sont pas retenues car elles nécessitent un opérateur et sont propriétaires.
- SigFox, technologie radio conçue en 2009, non retenue car propriétaire ;
- LoRaWAN, technologie radio conçue en 2012, non retenue car propriétaire.

1. Chaque nœud, ou plus précisément chaque interface sur chaque nœud, a un identifiant lui permettant d'être contacté.

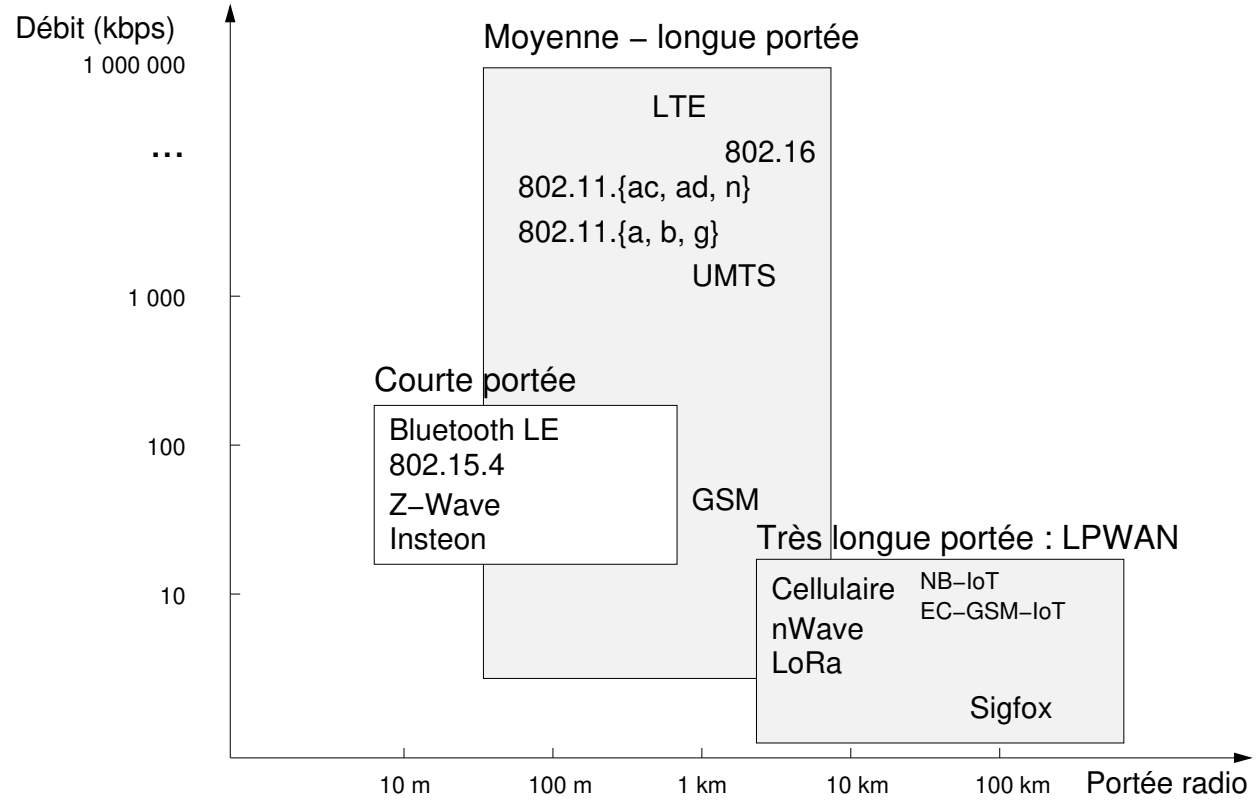


FIGURE 2.1 – Comparaison des technologies de communication (couches physique et liaison) sous l'angle débit et portée.

Les technologies que nous catégorisons « de moyenne portée » (500 m à quelques kilomètres) ont pour objectifs de maximiser à la fois le débit et la portée. Le débit élevé indique un manque d'économie d'énergie, ce qui va à l'encontre de ce qui est souvent recherché dans un réseau IoT. Parmi ces technologies, nous pouvons citer :

— Les technologies cellulaires (GSM, UMTS, LTE) et leurs variantes intermédiaires : leur objectif initial était de fournir suffisamment de bande passante pour la voix et de simples messages, puis les technologies se sont orientées vers l'accès à Internet.
— WiMax : son objectif est de fournir un accès à Internet à des particuliers, le débit recherché est de l'ordre de plusieurs dizaines de Mbps.
— 802.11, et ses multiples variantes : son objectif est de fournir un débit intéressant pour l'accès Internet à des particuliers. Le débit est un critère prioritaire ici également.

Enfin, les technologies de courte portée ($\leq$ 1 km), faible débit, avec un matériel généralement peu onéreux et suffisant pour les usages attendus (par exemple échanger des messages à cent mètres, économiser de l'énergie pour fonctionner sur batterie). Parmi ces technologies, nous pouvons citer :

— Bluetooth Low Energy : la technologie Bluetooth est composée de dizaines de profils différents, qui donnent lieu à des piles de communication différentes. Bluetooth Low Energy est une variante de Bluetooth, avec une pile de communication simplifiée pour limiter la consommation énergétique.
— Z-Wave, technologie conçue en 1999, non retenue car propriétaire ;
— Insteon, technologie conçue en 2005 : la charge utile du protocole Insteon est de 24 octets, ce qui est insuffisant pour les protocoles de plus haut niveau et une protection des communications indépendante de la couche liaison. Cette technologie est donc non retenue car elle ne laisse pas de place aux protocoles de niveaux supérieurs.

— Ant, technologie conçue en 2013 et 2014 (ANT+), non retenue car propriétaire : au delà de l'aspect propriétaire, les paquets ont seulement 8 octets de charge utile.
— IEEE 802.15.4

Pour les couches réseau physique et liaison, notre étude se focalisera sur IEEE 802.15.4. Cette technologie est un standard IEEE [11], avec de multiples mises à jour. IEEE 802.15.4 est utilisé comme couche liaison dans des projets de grande envergure (comme WirelessHart, 6LoWPAN, Zigbee ou Thread), et est fortement déployé (plus de 500 millions de puces vendues). De plus, cette technologie a fait l'objet de recherches académiques [12] [13], nous permettant d'avoir du recul.

2.2.3 IEEE 802.15.4

Plusieurs aspects de IEEE 802.15.4 nous intéressent pour notre étude, à savoir le fonctionnement de la radio, le temps d'échange attendu, et les limitations sur les paquets envoyés.

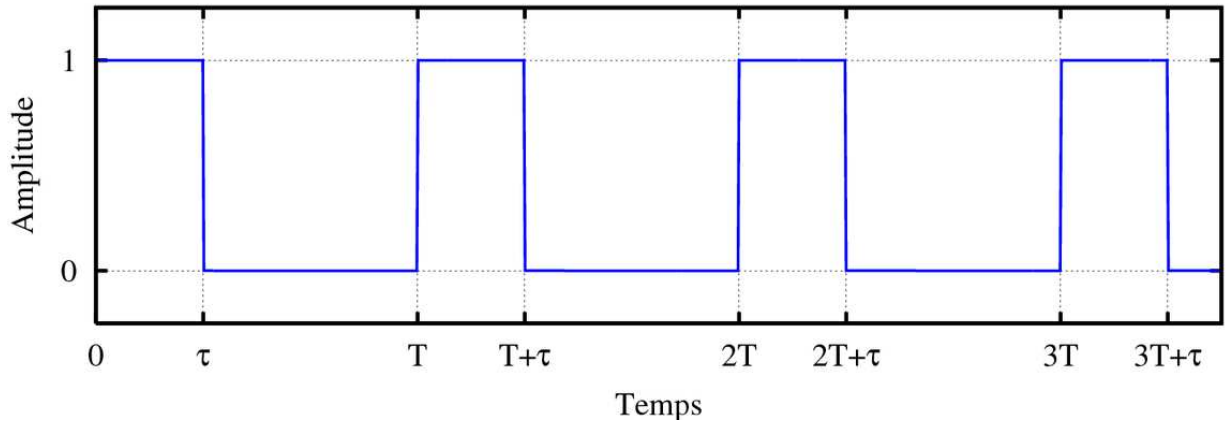


FIGURE 2.2 – Fonctionnement du rapport cyclique. Pour une radio, nous pouvons considérer l'amplitude de 1 comme une radio activée, 0 comme une radio éteinte. Le rapport cyclique est ici de $\frac{t}{T}$, avec t l'intervalle de temps pendant laquelle la radio est activée et T la période. (extrait de Wikipédia)

L'accès au médium peut se faire de deux manières : soit par allocation de plages de temps, soit par contention avec des techniques d'évitement de collisions. Dans le premier cas, les nœuds ont besoin d'horloges synchronisées. Chaque nœud se réveille lorsqu'il doit envoyer une donnée et une gestion est faite, entre autres, pour réveiller les nœuds intermédiaires lorsque le réseau est étendu sur plusieurs sauts. Dans le second cas, les nœuds doivent écouter le réseau régulièrement pour savoir si un message est transmis, et l'envoi d'un message est géré par un algorithme d'évitement de collisions comme CSMA-CA. L'idée générale derrière CSMA-CA est de vérifier que le médium n'est pas occupé avant de transmettre un message, et si le médium est occupé alors le nœud attend un certain temps avant un nouvel essai [2]. Dans notre environnement contraint, les objets n'échangent pas des données de manière régulière, les échanges sont conditionnés par des requêtes irrégulières et imprévisibles de clients extérieurs au réseau. Par conséquent, l'accès au médium se fera via une contention du réseau, avec une technique de CSMA-CA.

La puce radio peut rester active pour écouter le réseau en continu, ou être activée par intermittence, pour écouter le réseau à intervalles réguliers. Cette intermittence est nommée *rapport cyclique*, comme le montre la figure 2.2. Avec un rapport cyclique, les nœuds activent leur radio à intervalle régulier pour savoir si un nœud souhaite communiquer avec eux. La durée d'activation de la radio pour la détection de message est minime. Sans message à transmettre ou à recevoir, la radio est alors éteinte la majeure partie du temps. Sachant que la radio est une source majeure de consommation d'énergie

2. Le temps d'attente avant un nouvel essai est aléatoire, pour diminuer le risque de collision entre deux nœuds, et ce temps augmente avec le nombre d'essais.

sur un micro-contrôleur [14], le gain d'énergie apporté est intéressant. Ce fonctionnement a tout de même un coût de synchronisation : la radio est activée régulièrement pour détecter la transmission de messages, mais pour ne pas perdre une partie de la donnée transmise il faut ajouter un préambule, c'est-à-dire une suite reconnaissable d'octets indiquant le début d'un message en transmission. La durée de transfert de ce préambule doit être suffisamment longue pour être détectée à temps. Pour éviter tout biais engendré par un mauvais usage de la radio, notre environnement cible est composé d'objets avec une radio toujours activée, ce qui est envisageable notamment dans l'industrie 4.0.

Pour expliquer le temps pris par un paquet pour arriver à destination, nous devons connaître la durée d'attente avant une transmission. Cette information est disponible dans le standard IEEE 802.15.4 [11]. Le temps d'attente pour CSMA-CA (exprimé en symboles) est décrit par l'équation suivante :

$$T_{\text{attente}_{\text{symboles}}} = \text{aléa}(2^{\text{BE}} - 1) \times \text{unité de durée d'attente} \tag{2.1}$$

BE signifie *Backoff Exponent*, ce chiffre est utilisé pour définir une limite haute au temps d'attente dans l'algorithme CSMA-CA. L'*unité de durée d'attente* correspond à une durée d'attente avant envoi d'un message, utilisée comme base pour la durée d'attente effective qui sera un multiple de cette durée. Cette durée est une constante fixée à 20 dans le standard IEEE 802.15.4.

Cette équation mène au temps maximum d'attente pour le CSMA-CA :

$$\begin{aligned} T_{\text{attente}} &= \frac{T_{\text{attente}_{\text{symboles}}}}{\text{nombre de symboles par seconde}} \\ &= \frac{\text{aléa}(2^3 - 1) \times 20}{62\,500} \\ &\leq 2,24 \text{ ms} \end{aligned} \tag{2.2}$$

Le temps d'attente maximum au premier essai lorsque nous avons un message à envoyer est de 2,24 ms. Dans le cas le plus favorable, c'est-à-dire à la première attente pour accéder au médium, *BE* vaut 3. Si la fonction aléatoire suit une distribution linéaire, alors le temps d'attente moyen sera de 1,12 ms avant d'envoyer un message.

		Octets		
		1		**variable**
Preamble	SFD	Frame length (7 bits)	Reserved (1 bit)	PSDU
SHR		PHR		PHY payload

FIGURE 2.3 – Couche physique du standard IEEE 802.15.4 2006. Le préambule fait 4 octets et le SFD est de 1 octet. *Extrait de [11].*

Enfin, la technologie de niveau 2 utilisée implique une limitation sur la taille des paquets, ceux-ci ne peuvent dépasser 127 octets. Les figures 2.3 et 2.4 détaillent les en-têtes des messages selon le standard IEEE 802.15.4. Dans le reste de notre étude, nous utilisons de 11 à 23 octets pour les en-têtes de la couche liaison, soit 2 ou 8 octets par adresse (selon ce qu'il est possible d'utiliser dans l'architecture étudiée). Nous utilisons 2 octets pour l'adresse Personal Adress Network (PAN) de destination, qui permet de séparer des réseaux IEEE 802.15.4. Nous n'utilisons pas d'adresse PAN source car nous considérons que les nœuds communiquent uniquement dans leur réseau. Enfin, nous n'utilisons pas de sécurité de niveau 2, ce qui ajouterait jusqu'à 14 octets. Dans les paquets IEEE 802.15.4, il ne reste

Octets: 2	1	0/2	0/2/8	0/2	0/2/8	0/5/6/10/14	variable	2
Frame Control	Sequence Number	Destination PAN Identifier	Destination Address	Source PAN Identifier	Source Address	Auxiliary Security Header	Frame Payload	FCS
		Addressing fields						
MHR							MAC Payload	MFR

FIGURE 2.4 – Couche liaison du standard IEEE 802.15.4 2006, qui correspond au champ PSDU de la couche physique. Dans notre environnement, l'adresse PAN de destination fait 2 octets, et il n'y a pas d'adresse PAN source ni d'en-tête de sécurité. Selon l'architecture, des adresses de 2 ou 8 octets sont utilisées. Les messages envoyés en broadcast ont une adresse de destination de 2 octets. *Extrait de [11].*

que 116 octets disponibles pour les en-têtes des protocoles de plus haut niveau et pour la charge utile, voire 104 si des adresses de 8 octets sont utilisées.

2.2.4 Conclusion

Les couches physique et liaison cibles de nos travaux sont IEEE 802.15.4, qui est un standard déjà fortement déployé et dont la principale limitation pour nos travaux est la taille maximale des paquets (127 octets). La communication sans fils a un accès par contention avec un algorithme d'évitement de collisions (CSMA-CA) et nous considérons la radio toujours allumée.

2.3 Couche réseau

Nous souhaitons un environnement cible qui accepte les requêtes de clients sur Internet, ce qui correspond à une architecture IoT de référence, et pour cela nous utilisons un adressage IPv6 via 6LoWPAN.

2.3.1 6LoWPAN

IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) [15] [16] est une introduction de IPv6 dans les réseaux contraints. 6LoWPAN permet à un routeur de diminuer la taille des en-têtes IP et UDP dans un réseau contraint, les adresses IP utilisées sont donc différentes (plus courtes) que celles utilisées sur un réseau IP non contraint, mais les deux sont compatibles. Les adresses utilisées sont dérivées des adresses de couche liaison de 802.15.4, permettant à 6LoWPAN d'omettre complètement les adresses de l'en-tête IPv6 (car déduite de l'adresse MAC). L'en-tête IP est ainsi réduit jusqu'à 2 octets, et à 20 octets dans le cas le plus défavorable (préfixe du client non connu à l'avance). 6LoWPAN implique une passerelle spécialisée, qui traduit les paquets pour le réseau contraint, appelée « routeur de bordure 6LoWPAN ».

	même réseau	réseaux différents
aucune	114	94
UDP	110	90
TCP	94	74

TABLE 2.2 – Charge utile disponible pour la couche applicative, selon le protocole de transport utilisé et la provenance de la requête sur le réseau 6LoWPAN, sur des liens IEEE 802.15.4. La taille de l'en-tête de la couche liaison est de 11 octets.

Les tailles des en-têtes 6LoWPAN varient dans les messages échangés selon plusieurs critères. Le tableau 2.2 résume la taille de charge utile restante d'un paquet dans un réseau IEEE 802.15.4, 6LoWPAN, UDP et CoAP. Ce tableau prend en compte un en-tête liaison de 11 octets [3], des protocoles de transport différents (aucun, UDP avec optimisation 6LoWPAN et TCP) et le contexte de la communication (appareils dans le même réseau, ou dans des réseaux différents). Par exemple, pour la communication entre deux objets du même réseau, les préfixes d'origine et de destination peuvent être retirés : les adresses de la couche liaison suffisent à identifier les nœuds, l'en-tête IP est donc réduit à 2 octets. Si le message provient d'un client, et que le préfixe est connu par l'objet destinataire, alors le préfixe de ce client n'est pas envoyé : l'en-tête IP est alors de 12 octets (2 octets de couche réseau, 1 octet de contexte, 1 octet hop-limit, et 8 octets d'adresse source), 20 si le préfixe du client n'est pas connu.

Plusieurs optimisations sur la taille des messages existent et sont liées à 6LoWPAN :

— une version tronquée de UDP, avec un en-tête de 4 octets (mais toujours 20 octets pour TCP) ;

— 6LoWPAN-GHC [17], un mécanisme qui utilise un algorithme de compression pour diminuer la taille des en-têtes ;
6LoWPAN-GHC implique des calculs supplémentaires et nécessite de la mémoire vive pour réduire la taille des messages. La compression est efficace même quand un message est chiffré avec DTLS puisqu'il supprime une redondance dans l'en-tête DTLS.

— le partage de « contextes », mécanisme consistant à échanger des préfixes réseau utilisés par les clients, afin de ne pas les transmettre par la suite dans les messages.

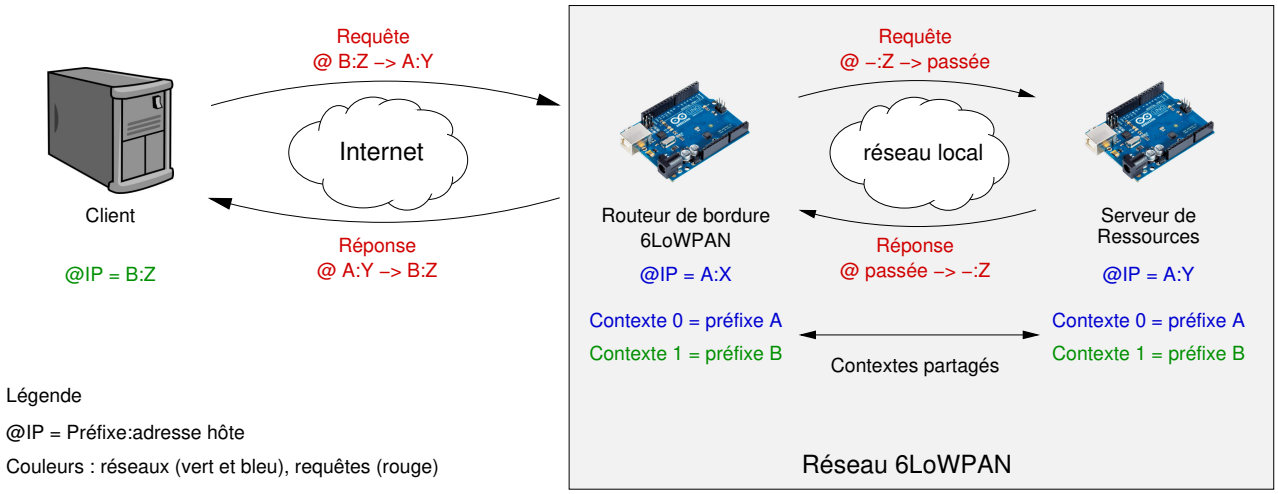


FIGURE 2.5 – Partage de contextes 6LoWPAN

Dans un environnement 6LoWPAN les adresses IPv6 peuvent être tronquées si le routeur et l'objet partagent des contextes (c'est-à-dire des préfixes). La figure 2.5 représente le mécanisme de partage de contextes 6LoWPAN. En premier lieu, il y a un échange des préfixes utilisés par les clients ; cet échange n'est pas standardisé au moment où ces lignes sont écrites. Ensuite, lors d'un échange avec le client, le routeur omet le préfixe du client (indiqué par un trait), réduisant la taille du message. De plus, l'objet dans cet exemple est à un saut du routeur, l'adresse IP de l'objet est supprimée de la requête par le routeur car l'adresse MAC de destination est utilisée comme identifiant (l'adresse est dite *passée* dans le schéma). Il en est de même pour la réponse de l'objet qui ne contient pas l'adresse IP source puisque celle-ci est déduite de l'adresse MAC. Le détail des tailles et des contenus des messages est en annexe B.

3. Les 11 octets de l'en-tête liaison correspondent à des adresses liaison de 2 octets et 2 octets d'adresse PAN, soient les paramètres cibles de la couche liaison, établis à la section précédente.

2.3.2 Routing Protocol for Low-Power and Lossy Networks (RPL)

Dans ce document nous ferons référence au protocole IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) [18] qui sera utilisé conjointement avec 6LoWPAN.

RPL est un protocole de routage pour des réseaux contraints sans fil, peu fiables (qui ont des pertes de paquets) et une connectivité intermittente. L'objectif est d'être résilient, c'est-à-dire d'optimiser l'accessibilité des objets en recréant de nouveaux chemins lorsqu'un nœud intermédiaire devient inaccessible. En premier lieu, le routeur distribue des informations aux nœuds (comme le préfixe du réseau par exemple), afin que les nœuds construisent leurs adresses locale et globale. Ensuite, le routeur génère puis maintient un graphe acyclique dirigé vers la destination (Destination-Oriented Directed Acyclic Graph), qui représente les chemins vers les nœuds, ce qui correspond à sa table de routage.

2.3.3 Conclusion

La couche réseau utilisée dans nos travaux est IPv6 via 6LoWPAN, ce qui permet une connectivité de bout-en-bout entre les objets et des clients sur Internet. 6LoWPAN est conçu pour les environnements contraints et ajoute un en-tête réduit pour diminuer l'empreinte de IPv6 dans les messages.

2.4 Couche applicative

Dans cette section, nous expliquons notre choix de CoAP comme couche applicative dans notre étude. En effet, deux protocoles sont principalement utilisés pour la couche applicative dans les réseaux contraints : MQTT-SN et CoAP.

2.4.1 MQTT-SN

MQTT-SN (anciennement pour « Message Queuing Telemetry Transfer Sensor Networks ») [19] est un protocole basé sur le mécanisme de publication et souscription (*publish-subscribe*) standardisée par l'organisme OASIS. Le protocole permet à des capteurs dans un réseau contraint d'envoyer régulièrement des mises à jour des données à un routeur. La récupération d'une donnée par un client peut être découpée en trois temps :

1. un client s'inscrit auprès du routeur pour recevoir des mises à jour sur un sujet (c'est-à-dire une information venant d'un capteur) ;
2. un objet envoie une donnée au routeur, correspondant à une mise à jour de la valeur ;
3. le routeur transmet la donnée au client.

MQTT-SN a une faible taille d'en-têtes et une gestion de la fiabilité (MQTT-SN permet de vérifier qu'un message a bien été reçu). Ce protocole a cependant un défaut majeur : l'envoi de commandes sur les objets contraints n'est pas pris en compte.

MQTT-SN correspond au comportement prévu dans une architecture maître-esclave car le client ne contacte pas directement l'objet. Cependant, le manque de prise en charge des actionneurs limite le protocole à une application de récolte de données. Pour cette raison, MQTT-SN ne fait pas partie de l'environnement cible de ce document, et ne sera pas utilisé dans une comparaison avec une architecture de référence.

2.4.2 Constrained Application Protocol

Constrained Application Protocol (CoAP) [20] est un protocole applicatif conçu par le groupe de travail IETF CoRE pour les environnements contraints. Ce protocole a une forte similarité avec le protocole HTTP, ils ont la même sémantique, ce qui permet une bonne intégration au web. CoAP est pensé pour des réseaux avec un faible MTU, la taille de l'en-tête minimale étant de 4 octets. CoAP

est également adapté aux objets avec une faible capacité de calcul, les en-têtes sont binaires et leur analyse est simple pour un microcontrôleur.

Au niveau des fonctionnalités, CoAP prend en charge :

— un serveur mandataire ;
— un service de cache de données ;
— la découverte de ressources : elle est effectuée via le mécanisme REsource LOcation and Discovery RELOAD [21], où les objets annoncent leurs ressources à des serveurs (et peuvent mettre en cache des valeurs). La découverte de ressources n'est pas utilisée par la suite, puisque cette fonctionnalité est hors du cadre de ce document.
— la souscription à une ressource, via le mécanisme d'observation : un client peut demander à un objet de lui transmettre les mises à jour d'une donnée lorsque celle-ci change de valeur. Ce mécanisme contribue à la diminution du nombre de messages, l'envoi d'une mise à jour étant effectué seulement lors d'un changement de la valeur mesurée et sans nécessiter de requêtes régulières. La différence avec MQTT-SN est que l'objet est sollicité par un client et il n'envoie pas systématiquement ses mises à jour à une passerelle.
— la fiabilité : CoAP a deux types de messages, le message CON (pour *confirmable*) qui doit être acquitté, et NON (pour *non confirmable*) qui n'a pas besoin d'être acquitté. CoAP permet donc de vérifier qu'un message a bien été reçu.
— la communication de groupe : CoAP permet une communication multicast, ce qui simplifie la récupération de données sur un ensemble d'objets. Une requête seule suffit à récupérer des données sur plusieurs objets ou leur envoyer une commande ;
— l'intégration aux API web [22] : l'intégration avec des API web est un objectif important pour le protocole CoAP. Pour cela, toute ressource est vue comme une ressource REST, sur laquelle nous pouvons utiliser les verbes habituels de HTTP, à savoir `GET`, `POST`, `PUT` et `DELETE`. Ainsi, une ressource sur un nœud A est accessible via une URL, par exemple `coap://exemple.com/A/température`.

De plus, dans la littérature, CoAP se base sur 6LoWPAN pour les couches réseau et transport. Ces deux protocoles sont donc adaptés pour travailler de pair.

Enfin, des travaux sont en cours pour intégrer un mécanisme de publication et souscription à CoAP [23], comme celui de MQTT-SN. Ce mécanisme permet d'effectuer des mises à jour lors d'un changement d'une valeur d'un capteur sans nécessiter de requêtes auprès des objets. La différence avec le mécanisme d'observation est que les données sont envoyées à la passerelle (ou une autre machine) et pas directement à chaque client qui ferait la demande de la donnée. Cette extension permettrait aux objets les plus contraints, d'un point de vue énergétique, de dormir aussi longtemps que possible. Contrairement à MQTT-SN, les clients peuvent envoyer des commandes aux objets.

2.4.3 Conclusion

Nous considérons CoAP comme couche applicative pour les environnements contraints. Ce protocole est adapté à ces environnements grâce à son en-tête court, sa communication de groupe et son mécanisme d'observation. CoAP est également adapté à une intégration aux applications web grâce à son API compatible avec HTTP. De plus, CoAP permet de demander des informations à des capteurs et envoyer des commandes aux actionneurs ; ce protocole est donc pertinent dans les réseaux avec des capteurs et des actionneurs.

2.5 Architecture

Une architecture est une organisation des différents éléments du système, logiciels et matériels, et les relations entre ces éléments. La figure 2.6 illustre une architecture minimaliste composée de deux périphériques, un client et un objet contraint, qui communiquent via le protocole CoAP dans un

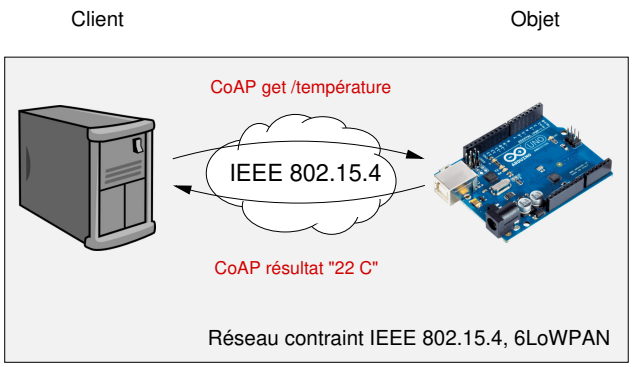


FIGURE 2.6 – Architecture exemple.

réseau IEEE 802.15.4 et IPv6 (6LoWPAN). L'architecture inclut les périphériques et leurs interactions, c'est-à-dire quels sont les périphériques qui communiquent et les protocoles utilisés.

2.6 Résumé de notre cible

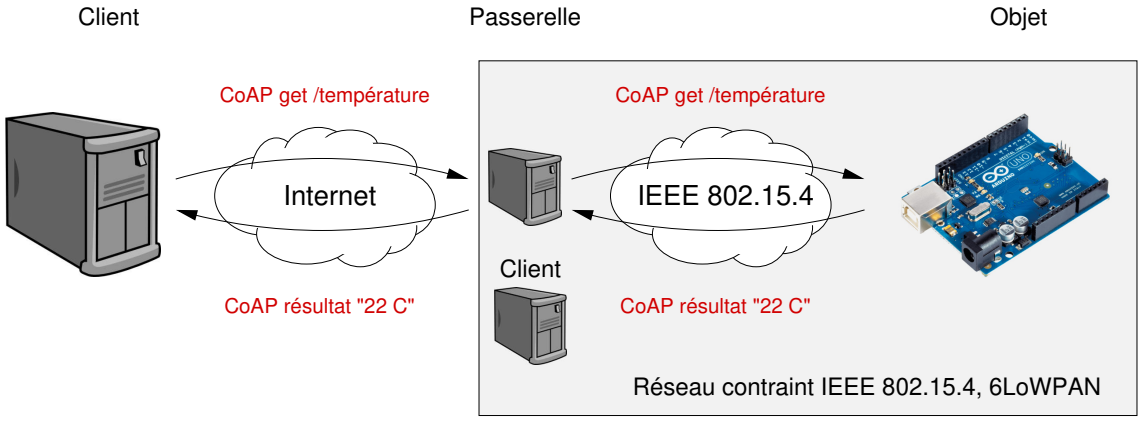


FIGURE 2.7 – Architecture cible.

Dans le reste de notre étude, nous nous focaliserons principalement sur un ensemble de technologies qui forment un environnement contraint standard, avec des technologies ouvertes et répandues.

Premièrement, d'un point de vue du matériel, les objets sont de classe 1 selon 2.1, avec une fréquence d'horloge du micro-contrôleur autour de 16 MHz. Les objets n'ont pas de matériel spécifique pour accélérer les calculs cryptographiques.

Ensuite, la communication dans le réseau contraint se passe sur des liaisons IEEE 802.15.4, la radio est toujours activée, et l'accès au médium est géré via l'algorithme CSMA-CA. De cette manière, nous utilisons un protocole de communication standard et ouvert, sans biais lié à une radio éteinte au mauvais moment, et avec un accès au médium qui ne pénalise pas les protocoles de communication ayant de nombreux messages.

La couche réseau est basée sur IPv6 pour avoir une accessibilité des objets sur Internet grâce à un adressage IP, comme dans une architecture de référence. 6LoWPAN est utilisé pour limiter la taille des en-têtes IP et UDP dans les environnements contraints que nous étudierons.

Enfin, la couche applicative CoAP est utilisée, car le protocole est adapté aux objets contraints de par ses en-têtes courts et facilement analysables. CoAP permet de récupérer des données ou d'envoyer des commandes, ce qui permet de contrôler à la fois des capteurs et des actionneurs. De plus, CoAP

permet une intégration simple aux standards du web (REST, traduction simple des requêtes HTTP vers CoAP), ce qui facilite l'adoption de cette technologie.

La figure 2.7 illustre notre architecture cible. Dans notre architecture, des clients accèdent à des informations ou envoient des commandes à des objets dans un réseau contraint. Ces clients sont internes ou externes à ce réseau. Le réseau est basé sur les technologies IEEE 802.15.4 et 6LoWPAN, et le protocole applicatif est CoAP (lorsque cela est possible).

2.7 Conclusion

Nous avons détaillé une architecture qui dispose de matériels contraints (10 Kio RAM, 100 Kio Flash, 16 MHz, alimentation sur secteur) et reprend les technologies IEEE 802.15.4, 6LoWPAN et CoAP comme les mécanismes composant notre architecture de référence. Cette architecture est composée d'objets dans un réseau local IEEE 802.15.4, accessibles via une passerelle (sans contraintes) par des clients sur Internet ou dans le réseau local. Elle est représentative d'un réseau IoT et sera utilisée comme telle dans la suite de notre document.

Dans le chapitre suivant, nous allons compléter cette présentation avec la protection des communications en prenant en compte les contraintes évoquées de mémoire (RAM et code), de temps de calcul et de communication.

Chapitre 3

Sécurité

Notre étude consiste à analyser ce qu'apporte une architecture de sécurité maître-esclave dans l'IoT, ainsi que ses implications d'un point de vue de la sécurité et des contraintes sur les matériels. Nous analysons également la protection des communications sur un matériel contraint. Ce chapitre présente les éléments nécessaires à la protection des communications dans une architecture.

Premièrement dans ce chapitre, nous posons les bases de la sécurité des communications. Nous montrons ensuite les spécificités des objets contraints en matière de sécurité, ainsi que les algorithmes de protection des communications dans les réseaux contraints. Afin de mieux situer notre travail, ce chapitre comporte également une description des attaques possibles dans ce type de réseaux et celles que nous souhaitons contrer. Enfin, nous détaillons les différentes étapes de déploiement.

Ce chapitre présente les points suivants :

1. Les principes de protection des communications et les algorithmes associés (authentification des pairs, autorisation, protection des messages) ;
2. La sécurité des communications dans les environnements contraints (algorithmes adaptés, attaques dans ces réseaux, phases de déploiement).

3.1 Communication sécurisée : principes généraux

Dans cette section, nous introduisons la sécurisation des communications, c'est-à-dire la confidentialité des échanges (chiffrer, vérifier l'intégrité et l'authenticité des messages) et la confiance entre les participants (authentification, partage de clés).

3.1.1 Protection des messages

La confidentialité d'une communication nécessite le chiffrement des messages. L'objectif du chiffrement est de rendre un texte en clair incompréhensible pour quiconque n'ayant pas le droit d'accéder à la donnée. Le fonctionnement est le suivant :

1. prendre un texte clair en entrée (et éventuellement une suite d'octets aléatoires nommée *vecteur d'initialisation*) ;
2. modifier le texte pour le rendre incompréhensible, via une clé de chiffrement. Quiconque possède la clé peut rétablir le texte dans son état original, l'opération est dite *réversible*.

L'algorithme Rijndael est un exemple d'algorithme de chiffrement, plus connu sous le nom d'*Advanced Encryption System* (AES).

Le chiffrement d'un message n'est pas suffisant, il faut également s'assurer qu'il vient du bon interlocuteur, et qu'il n'a pas été modifié durant le trajet. Les algorithmes de hachage sont une solution à ces deux problématiques : l'idée est de créer une empreinte d'une donnée (*Integrity Check Value*, ICV

[24]). Deux informations différentes doivent avoir une très haute probabilité de donner deux empreintes différentes, ce qui permet de vérifier l'intégrité d'une donnée. L'authenticité est vérifiée si l'empreinte est générée à partir d'un secret, pour que seul un possesseur du secret puisse générer l'empreinte (cette fois nommée *Message Authentication Code*, MAC). L'algorithme SHA256 est un exemple d'algorithme de hachage sans clé, mais qui peut être aussi utilisé en combinaison d'une clé (HMAC-SHA256, pour *keyed-Hash Message Authentication Code*).

3.1.2 Modes de chiffrement

La protection d'une donnée passe par le choix du mode de chiffrement du flot de données. Une donnée ne peut pas être sécurisée en chiffrant simplement des blocs un à un avec un algorithme de chiffrement et une clé car une même donnée chiffrée donnera toujours le même résultat, conservant des motifs reconnaissables (voir le mode ECB, annexe A). Une méthode pour obtenir une donnée chiffrée plus diffuse, moins reconnaissable, est de réutiliser le résultat du chiffrement d'un bloc précédent pour chiffrer le suivant (voir le mode CBC, annexe A).

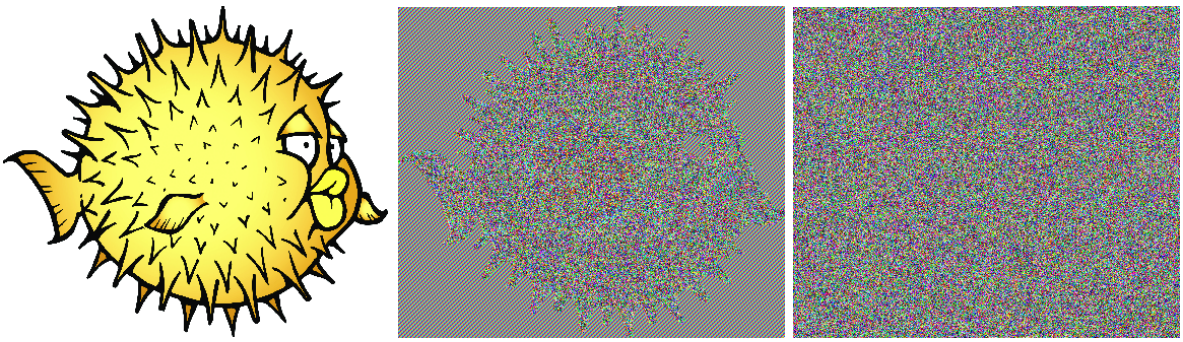


FIGURE 3.1 – Image originale, chiffrée simplement bloc par bloc (ECB), ou chiffrée de façon sécurisée (CBC).

La figure 3.1 illustre deux modes de chiffrement, ECB et CBC. La seconde image est le résultat d'un chiffrement bloc par bloc le plus simple possible, soit le mode ECB : un motif de l'image originale se dessine, la silhouette du poisson apparaît. La troisième image est le résultat d'un chiffrement utilisant le résultat du chiffrement d'un bloc précédent pour chiffrer le suivant, ce qui correspond ici au mode CBC. De plus, ce mode utilise un compteur et une nonce (*number used once*, un nombre utilisé une seule fois) : une même donnée chiffrée plusieurs fois, y compris avec la même clé, donne deux résultats différents. Cette dernière image représente ce qui est attendu d'un chiffrement efficace, c'est-à-dire un résultat sans indice sur le contenu original, diffus, sans motif, sans recoupement statistique possible.

Les modes de chiffrement sont utilisés conjointement avec une méthode pour générer une empreinte de la donnée. Pour sécuriser plus rapidement un flot de données, des algorithmes ont été conçus pour ne nécessiter qu'une seule passe ; ils chiffrent et génèrent une empreinte en même temps. Ces algorithmes sont regroupés sous le terme générique « Authenticated Encryption with Associated Data » (AEAD), et utilisent une seule clé pour chiffrer, authentifier et vérifier l'intégrité des messages. Par exemple, l'algorithme « Counter with CBC-MAC » (CCM) utilise du chiffrement symétrique par bloc (par exemple AES), le mode CBC-MAC et un compteur (voir annexe A).

La protection d'une communication consiste à chiffrer, authentifier et vérifier l'intégrité des messages, mais il faut également s'assurer que l'on communique avec le bon interlocuteur.

3.1.3 Authentification des pairs

Une authentification est une vérification d'un attribut (par exemple une identité). Dans une communication, une machine A s'authentifie auprès d'une machine B, c'est-à-dire que A prouve sa possession de la clé en effectuant une opération cryptographique sur une donnée connue de B. L'authentification peut s'effectuer via un mécanisme de défi-réponse : pour s'authentifier, A effectue une opération cryptographique sur une donnée envoyée par B, B effectue la même opération et compare les résultats. Dans la pratique, plusieurs méthodes d'authentification existent et peuvent se combiner.

La première méthode d'authentification nécessite des clés *symétriques*, comme un mot de passe ou une clé pré-chargée (PSK), qui sera utilisé(e) pour générer la preuve de possession de la clé, via un algorithme de chiffrement ou de hachage. Ces clés sont utilisées pour le chiffrement, mais également pour le déchiffrement.

Une seconde méthode consiste à utiliser des clés *asymétriques* :

— des clés brutes, c'est-à-dire deux paires de clés publiques et privées (une paire pour chaque interlocuteur) ;
Les clés publiques sont connues à l'avance (pré-chargées ou transférées par un moyen sûr dans les appareils), impliquant une confiance entre les deux appareils. Chaque interlocuteur possède une clé privée servant à signer un message, et une clé publique permettant de vérifier cette signature : par exemple, A signe un message avec sa clé privée et B utilise la clé publique de A pour vérifier la signature, B est alors assuré que le message vient de A.

— des certificats, signés par un tiers connu (vérifiable et de confiance) ;
Les certificats sont utilisés en combinaison d'une paire de clés publique et privée (comme l'option précédente). Ils permettent de limiter la durée de validité d'une paire de clé, obligeant un renouvellement régulier du matériel cryptographique. Les certificats permettent également une authentification de deux interlocuteurs qui ne partagent pas de secret entre eux, grâce à la confiance qu'ils ont tous les deux dans un tiers connu (ou un même tiers).

— des tickets, signés par un tiers connu (vérifiable et de confiance).
Un ticket est une représentation d'autorisations (droits accordés à un client), avec durée de validité limitée.

D'autres mécanismes d'authentification existent avec des modèles de confiance différents comme PGP et la toile de confiance [25], mais sont hors du cadre de ce document.

L'authentification d'un appareil permet de vérifier un attribut (comme son identité, ou l'appartenance à un groupe), mais n'indique pas quels sont les droits de cet appareil.

3.1.4 Autorisation

Le mécanisme d'autorisation consiste à donner le droit à un pair d'accéder à une ressource. Dans ce document nous divisons l'autorisation en deux catégories, *implicite* et *explicite*. Une autorisation implicite est un contrôle des ressources à gros grain : aucun lien n'est établi entre la ressource et le client, chaque client (ou client authentifié) a accès à toutes les ressources d'un domaine sans autre condition. Une autorisation explicite implique qu'un mécanisme d'autorisation soit prévu, avec des échanges dédiés. De plus, un lien existe entre un ensemble de ressources et un ensemble de clients, permettant un contrôle précis des ressources. L'autorisation est parfois matérialisée par un échange d'une représentation de cette autorisation ; un ticket, par exemple, peut contenir les droits accordés à un client.

3.1.5 Partage de clés

La gestion de clés fait référence à l'établissement du matériel cryptographique utilisé pour sécuriser les communications. Elle est considérée automatique quand une clé cryptographique de court terme est générée pour une session, dérivée d'une clé de long terme (comme un secret partagé, qui n'est pas ou peu changé sur les objets).

Deux méthodes permettent de partager une clé entre deux interlocuteurs. La première consiste à utiliser un secret partagé, c'est-à-dire un secret connu des interlocuteurs mais sans avoir été échangé sur un canal public. Le secret est utilisé en combinaison d'un aléa partagé publiquement pour générer une clé *maître*. Cette clé est ensuite utilisée pour dériver des clés utilisées pour la protection de la communication. Une seconde méthode consiste à utiliser des algorithmes prévus pour le partage de clés, basés sur l'usage de cryptographie asymétrique, comme Diffie-Hellman (DH) et Diffie-Hellman Ephemeral (DHE). Les bonnes pratiques liées au partage de clés sont discutées dans [26].

Si un attaquant obtient le secret partagé, il peut rejouer les échanges passés et recréer les clés de chiffrement utilisées, puis déchiffrer les communications. Pour prévenir ce type d'attaque, certaines méthodes de partage de clés ont la propriété *Perfect Forward Secrecy* (PFS). Selon ce principe la découverte d'une clé de chiffrement d'une session n'implique pas la connaissance des messages envoyés dans des sessions précédentes. Une méthode simple pour respecter le principe de PFS serait que les interlocuteurs s'accordent sur une nouvelle clé générée aléatoirement à chaque début de session, et sur l'utilisation d'un secret partagé pour authentifier les échanges liés au partage de clés. La clé utilisée pour le chiffrement n'étant ni dérivée du secret partagé, ni stockée après la fin de la session, il n'est pas possible de déchiffrer la session.

3.1.6 Courbes Elliptiques (ECC)

Les courbes elliptiques (ou *Elliptic Curve Cryptography*, ECC) sont une optimisation des algorithmes asymétriques, utilisés principalement pour l'authentification et le partage de clés. Cette optimisation apporte deux avantages : les calculs sont un ordre de grandeur plus rapides par rapport à leurs équivalents sans optimisation, ce qui est un avantage peu importe le contexte, et les clés sont plus petites, permettant à des objets contraints en mémoire de stocker davantage de clés. Les courbes elliptiques sont très répandues, y compris dans les environnements non contraints, du fait de la vitesse d'exécution des algorithmes. Cette optimisation permet, par exemple, un plus grand nombre de connexions à la seconde pour un serveur web avec un trafic important. Les courbes elliptiques actuellement recommandées sont documentées dans [27].

La taille recommandée de clé, ainsi que les équivalences de taille de clés entre les algorithmes asymétriques, avec ou sans courbes elliptiques, et les algorithmes symétriques sont discutées dans [28]. Le National Institute of Standards and Technology (NIST) recommande une taille de clés d'au moins 224 bits pour des algorithmes à courbes elliptiques [29], ce qui rend impraticable l'authentification sur les nœuds contraints dans des temps raisonnables sans l'aide de matériel spécialisé.

Partager les clés de chiffrement avec un système de certificats est coûteux, même avec des courbes elliptiques [30]. Un appareil avec un processeur à 13 MHz prend plus d'une seconde pour initialiser, signer et vérifier un certificat avec des optimisations consommant environ 1500 octets de RAM, et 10 à 15 Ko de mémoire dédiée au code. L'exécution de ces trois opérations dure jusqu'à plusieurs secondes sur un processeur à 8 MHz. Par conséquent, l'usage de cryptographie asymétrique, y compris avec des courbes elliptiques, est réservée aux objets non contraints, contraints avec matériel spécialisé pour les calculs, ou encore contraints mais tolérant un temps d'authentification de plusieurs secondes (voire dizaines de secondes).

3.1.7 Chiffrement de bout-en-bout

Pour diminuer le risque d'une attaque sur les données (par exemple, une commande modifiée), il est préférable d'avoir à faire confiance au moins de nœuds possible dans le réseau. Si une donnée passe non chiffrée sur un nœud, il peut la modifier à sa guise et transmettre le message modifié. Par conséquent, moins il y a de nœuds qui voient le message en clair, moins grande est la surface d'attaque.

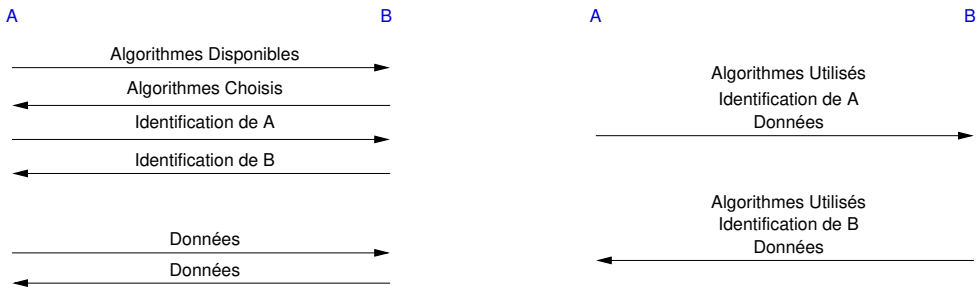
Le chiffrement de bout-en-bout est un principe selon lequel une seule et même session de chiffrement est utilisée, du périphérique qui acquiert la donnée (ou la génère) jusqu'au périphérique qui la demande. Par conséquent, aucun intermédiaire ne comprend le message. Sur Internet par exemple, lorsque nous

accédons à un page web sécurisée par HTTPS, seuls le navigateur et le serveur web comprennent la donnée échangée.

Ce principe est intéressant pour limiter la surface d'attaque : seuls le client et le périphérique qui acquiert la donnée ont besoin d'être sécurisés. La sécurité de bout-en-bout a cependant un coût : le périphérique qui acquiert la donnée doit garder une association de sécurité avec chaque client, signifiant une empreinte mémoire plus importante. Pour alléger cette contrainte, l'objet peut renouveler les associations de sécurité à chaque demande, échangeant la mémoire consommée contre plus de calculs et de messages.

3.1.8 Modes de communication

Les échanges entre deux interlocuteurs peuvent se faire en créant une session de communication maintenue, ou en utilisant un objet de sécurité.



(A) Mode de communication par sessions.

(B) Mode de communication par objets de sécurité.

FIGURE 3.2 – Modes de communication : session et objet de sécurité.

Une session de communication maintenue se crée via un échange préliminaire (une poignée de mains) avant les échanges de données. Cette poignée de mains consiste à négocier des algorithmes à utiliser pour la sécurité de la communication (pour l'authentification, le chiffrement, etc.), et à négocier le matériel cryptographique. La poignée de mains a pour avantage principal de diminuer la taille des messages, chaque message n'ayant pas à contenir tous les paramètres de session (comme par exemple les algorithmes utilisés). L'inconvénient est qu'il n'est pas possible d'utiliser un cache de données. Parmi les protocoles de sécurité connus et répandus qui utilisent une poignée de mains pour créer une session, nous pouvons citer TLS, DTLS, ou encore IPsec. La figure 3.2a illustre une session de communication entre deux appareils, A et B, et chaque flèche représente l'envoi d'un message. Dans cet exemple, A crée une communication sécurisée avec B en s'identifiant et en négociant des algorithmes, puis lui envoie une donnée dans un message séparé qui ne contient que la donnée.

Un objet de sécurité est un message sécurisé qui contient à la fois une indication sur les algorithmes utilisés, une négociation du matériel cryptographique, un moyen d'authentifier l'émetteur et enfin la donnée échangée. Les objets de sécurité ont pour avantage de réduire le nombre de messages, une poignée de mains n'étant plus nécessaire, rendant ainsi l'échange de données plus rapide sur un réseau avec un très faible rapport cyclique. De plus, les objets de sécurité facilitent la communication de groupe (via du multicast par exemple). Enfin, ils aident à répandre facilement l'usage de caches de données tout en appliquant du chiffrement de bout-en-bout. Les intermédiaires comme les serveurs mandataires et les entités de cache n'ont pas à déchiffrer le message pour fonctionner, mais peuvent quand même vérifier l'authenticité du message quand celui-ci est signé. L'inconvénient des objets de sécurité réside dans la taille des messages. La figure 3.2b illustre une communication par objets de sécurité entre deux appareils A et B. Dans cet exemple, seuls deux messages sont échangés, contenant à la fois l'identification des pairs, les algorithmes utilisés et les données.

Des procédures existent pour créer des objets de sécurité. Par exemple, CBOR Object Signing and Encryption (COSE) [31] est un ensemble de procédures pour signer, chiffrer et authentifier des messages, et utilisant des objets de sécurité. COSE utilise le format de données Concise Binary Object Representation (CBOR) [32] dont l'objectif est d'avoir un code et des messages légers. COSE est similaire à JSON Object Signing and Encryption (JOSE, 2013) [33] mais est spécifiquement conçu pour les environnements contraints [1].

Les deux modes de communication peuvent être complémentaires. Par exemple, une session de communication protège les communications entre un client et un cache données, et entre le cache et un objet. Dans cet exemple, les communications sont donc protégées, mais le cache voit les données en clair. Si des objets de sécurité sont utilisés pour protéger les informations, alors les données sont également protégées du cache de données (qui ne peut ni lire ni modifier les données). Bien qu'il ne comprenne pas les données, le cache accomplit son objectif en faisant correspondre une requête avec une réponse.

3.1.9 Suite cryptographique

Une suite cryptographique correspond à un ensemble d'algorithmes permettant de protéger une communication. Elle regroupe un algorithme de chiffrement, d'authentification et d'intégrité des messages, la taille des clés, la taille du MAC, le mode de chiffrement, ainsi que le protocole de partage de clé. Par exemple, la combinaison des algorithmes suivants forme une suite cryptographique : ECDHE (partage de clés), ECDSA (authentification), AES 128 bits (chiffrement), SHA256 (hachage), et CCM 64 bits (mode de chiffrement, d'authentification et d'intégrité des messages).

3.1.10 Autres mécanismes de protection des communications

Quelques propriétés des communications existent en marge de la cryptographie, très présentes dans les protocoles de communication et qui méritent une mention :

- la *protection contre le rejeu* : la protection contre le renvoi d'un message lu sur le réseau pour répéter une action, est toujours effectuée avec un numéro de séquence des messages, ce qui est simple et efficace.
- la *protection contre le déni de service* : empêche un serveur d'être surchargé en forçant l'initiateur de la connexion à effectuer une action, comme renvoyer une information (connu sous le nom de *cookie*) ou d'effectuer un calcul (un *puzzle*), avant que le serveur n'effectue de longs calculs ou ne retienne d'information à propos du client.
- le *chiffrement des métadonnées* : les architectures peuvent utiliser des protocoles qui chiffrent la couche applicative (par exemple CoAP) pour cacher les métadonnées (qui demande quoi et quand).
- l'*agilité cryptographique* : pour les déploiements hétérogènes et la migration d'une suite algorithmique à une autre, permettant des mises à jour vers un chiffrement plus fort [34].

Nous avons illustré les éléments importants de la protection d'une communication. Il nous reste à voir quelles sont les particularités des réseaux contraints, en terme d'algorithmes, d'attaques possibles et d'étapes de déploiement des nœuds contraints.

3.1.11 DTLS : protocole de sécurité

Un protocole de sécurité regroupe l'ensemble des mécanismes nécessaires à la communication sécurisée entre deux appareils. Datagram Transport Layer Security (DTLS) [35] est un protocole de sécurité, défini pour la dernière fois en 2012. Ce protocole fournit l'authentification des appareils par

1. Hors du contexte des environnements contraints, Cryptographic Message Syntax (CMS) est un autre exemple d'utilisation des objets de sécurité, utilisé par exemple dans la signature des emails. CMS est une notation standard et représentation de messages cryptographiques, utilisée pour signer, authentifier, résumer ou encore chiffrer des données arbitraires. CMS utilise le format ASN.1, qui limite l'usage de CMS dans les environnements contraints à cause de la taille des messages.

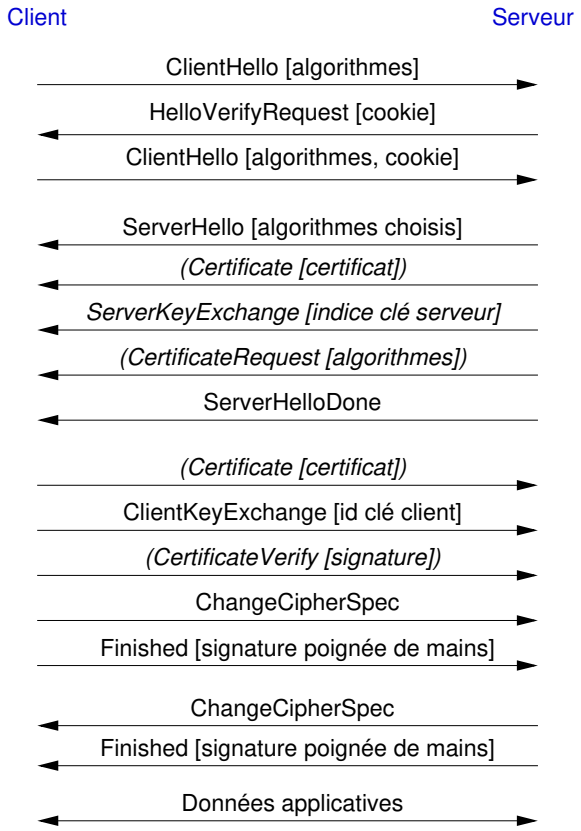


FIGURE 3.3 – Poignée de mains DTLS. Les messages en italique sont optionnels. La charge utile des messages est entre crochets. Les messages spécifiques à un contexte sont entre parenthèses.

différents mécanismes comme le secret partagé, les clés publiques ou encore via des certificats. Il fournit également la protection des communications (confidentialité, intégrité et authenticité des messages) avec par exemple des algorithmes AEAD comme CCM. DTLS gère le partage de clés, avec un secret partagé ou via l'algorithme Diffie-Hellman. Des profils d'usage pour l'Internet des Objets sont décrits par le groupe de travail IETF DICE [36]. De plus, DTLS inclut d'autres mécanismes, comme la protection contre le rejeu (via un numéro de séquence) et contre le déni de service (via l'échange d'un cookie).

Le mode de communication de DTLS repose sur des sessions. La figure 3.3 présente les 10 messages nécessaires pour l'établissement d'une connexion DTLS. Les deux premiers messages sont liés à l'échange d'un cookie, pour se protéger d'attaques par déni de service visant la mémoire des appareils. Les deux messages suivants forment la négociation des algorithmes, puis un message de signalisation est envoyé pour annoncer la fin de l'envoi de messages optionnels (illustrés en italique). Le client envoie ensuite son identité, puis deux messages, l'un indiquant le début du chiffrement, l'autre étant un message chiffré contenant une preuve de possession de la clé de chiffrement. Le serveur envoie les deux derniers messages de la poignée de main, qui ont le même objectif que les deux messages précédents, c'est-à-dire indiquer le début du chiffrement de la session et donner une preuve de la possession de la clé de chiffrement. Une fois la poignée de mains terminée, la session est ouverte et les données utiles sont échangées.

La figure 3.4 représente l'en-tête de DTLS lors de l'échange de données, qui est de 29 octets. Cet en-tête est composé de deux parties, la couche « Record » qui contient le type de message (CT), le numéro de séquence du message et sa taille, et la couche « Application » qui contient un nombre utilisé une fois (nonce) et la donnée échangée.

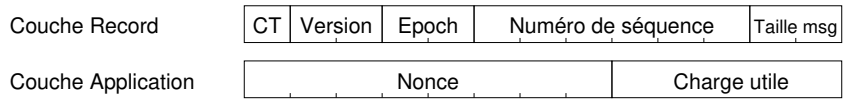


FIGURE 3.4 – En-tête DTLS.

Le protocole DTLS permet la négociation d'algorithmes cryptographiques (c'est-à-dire l'agilité cryptographique), et seuls 2 octets représentent une suite cryptographique ; il est donc possible de négocier un grand nombre d'algorithmes différents même au sein d'un réseau contraint.

Le nombre de message lors de l'établissement de la connexion retarde l'envoi de la première donnée. Ce problème peut être réduit en utilisant des connexions à long terme, c'est-à-dire en gardant la session ouverte même après la requête initiale. De plus, l'empreinte par message une fois la connexion établie est de 29 octets, ce qui limite la taille des données utiles sur des liens avec un faible MTU. Par exemple, sur une liaison IEEE 802.15.4, ces 29 octets représentent 23 % de la taille totale possible d'un paquet.

La sécurité de CoAP est basée sur DTLS, mais DTLS ne permet pas la communication de groupe sécurisée pour le moment [37]. De plus, la communication de groupe nécessite une gestion du multicast, qui n'est pas toujours pris en charge dans les réseaux pour des raisons de sécurité.

La section suivante présente la sécurité des communications face aux contraintes.

3.2 Sécurité dans les environnements contraints

La protection des communications est au cœur de notre travail. Dans cette section, nous indiquons les algorithmes utilisés dans les environnements contraints, selon les standards et les recommandations. Ensuite, nous illustrons les attaques possibles sur un réseau contraint, et les attaques couvertes par la protection des données (ce qui permet de cadrer le travail effectué). Enfin, un réseau contraint passe par plusieurs étapes (phases), de la configuration d'un nœud à sa mise hors service. Nous verrons donc également les étapes sur lesquelles nous allons nous concentrer dans la suite de ce document.

3.2.1 Algorithmes utilisés dans les environnements contraints

La RFC 7925 [36] émet des recommandations concernant les algorithmes à utiliser.

Les algorithmes symétriques sont privilégiés car plus rapides, comme AES pour le chiffrement par bloc. AES est un algorithme standard, répandu et obligatoire dans certains standards (comme 802.15.4 par exemple), ce qui implique une implémentation matérielle d'AES largement disponible sur les objets pour accélérer l'exécution. Une taille de clés de 128 bits est recommandée pour l'algorithme AES, c'est un compromis entre sécurité et vitesse d'exécution (128 étant la taille minimale actuellement considérée comme sécurisée par l'IETF et le NIST).

Les algorithmes AEAD, et en particulier CCM* [2], sont privilégiés comme mode de chiffrement, d'authentification et d'intégrité des messages. L'avantage est de ne nécessiter qu'un seul algorithme pour le chiffrement, l'authentification et la vérification de l'intégrité des messages. CCM* utilise un chiffrement par blocs (AES dans notre cas), et un algorithme de hachage SHA256. La RFC recommande également le secret partagé (*Pre-Shared Key*, PSK) comme méthode pour l'échange de clés et l'authentification.

Si les algorithmes asymétriques sont nécessaires, le document préconise l'usage de ECDH comme algorithme d'échange de clés, et ECDSA pour l'authentification. Pour le chiffrement, l'intégrité et l'authentification des messages, l'algorithme CCM* est toujours privilégié.

2. CCM* est réputé moins complexe à implémenter que CBC-MAC [38].

3.2.2 Attaques sur les réseaux contraints

Les réseaux contraints sont vulnérables à un certain nombre d'attaques [39], qui exploitent les limitations des matériels et du réseau. Les attaques et les moyens de les contrer sont explicités dans cette partie, classés par couche du modèle OSI. Notre travail devra couvrir les attaques au-dessus de la couche liaison et venant d'un client externe au réseau local.

3.2.2.1 La couche physique

Dans un réseau de capteurs, comme dans tout autre réseau sans fil, il est possible d'attaquer la disponibilité du réseau en créant des interférences (attaque *Brouillage*). Cette attaque peut se contrer avec du *Frequency Hopping Spread Spectrum* (FHSS), qui consiste à changer rapidement de canal de fréquence de communication en utilisant une séquence pseudo aléatoire connue des deux interlocuteurs.

Toujours sur la couche physique, un attaquant peut simplement récupérer ou détruire physiquement le capteur, dans le but de le compromettre ou de récupérer des informations contenues dans sa mémoire flash, ou simplement créer une indisponibilité par *dégradation* du matériel. Les ripostes nécessitent d'utiliser du matériel résistant ou de cacher le capteur, si cela est possible.

3.2.2.2 La couche liaison

Un attaquant peut créer un déni de service en créant volontairement des collisions. Les algorithmes de correction d'erreur deviennent inutiles, car au-delà de quelques bits d'erreurs il devient trop coûteux de prévenir ce genre d'attaques. Si le paquet est trop modifié durant sa transmission, il est ignoré.

Un attaquant peut également tenter de vider la batterie du capteur rapidement. Premièrement, le capteur peut essayer d'envoyer des données en vain, à cause d'un lien tombé ou de très mauvaise qualité. Un attaquant peut aussi envoyer des requêtes coûteuses en énergie au capteur. Dans ces deux cas il faut mettre une limite d'envoi (*rate limit*) sur le capteur pour limiter la perte d'énergie. Cette technique fonctionne si l'objet n'est pas compromis ; dans le cas contraire nous ne pouvons de toute manière pas faire confiance aux données échangées, et l'indisponibilité de l'objet est alors anecdotique.

Enfin, nous avons non pas une attaque au sens strict, mais une faiblesse au niveau du lien qui peut amener à un épuisement de la batterie ou à une mauvaise disponibilité : une attaque sur l'*équité*. L'attaquant perturbe le réseau en envoyant des messages longs et sans respecter les délais d'attente de CSMA-CA, ce qui est une forme amoindrie de déni de service causant des délais avant transmission allongés pour les messages légitimes.

3.2.2.3 La couche réseau

Un attaquant peut altérer ou *rejouer* (attaque *Rejeu*) une information telle qu'une information de routage ; ou encore usurper l'identité de celui qui émet pour envoyer lui-même des informations de routage factices. Ces attaques peuvent être contrées en ajoutant un code d'authentification de message au message de routage envoyé. De même, nous pouvons rajouter un compteur ou une estampille temporelle dans le message.

L'attaque *trou de vers* consiste à créer un lien avec une latence faible entre deux portions de réseau avec deux nœuds malicieux. Suivant le protocole employé dans le réseau, les nœuds sains peuvent choisir de faire transiter les messages par des nœuds malicieux ayant une meilleure latence pour atteindre le puits (c'est-à-dire le nœud où sont envoyées les données récoltées, qui peut être un routeur par exemple). Pour contrer ceci, il est possible d'ajouter des informations dans le paquet pour indiquer le lieu (si la position des capteurs est connue) où a été envoyé le paquet ou à quel moment, avec des estampilles. Une autre solution consisterait à authentifier des nœuds, pour ne choisir que des nœuds de notre réseau.

Si l'attaquant arrive à faire transiter les paquets par lui ou par un nœud corrompu, il peut alors faire du transfert sélectif de paquets. Pour prévenir ceci, nous pouvons établir un routage à plusieurs chemins.

Un nœud corrompu ou un attaquant pourrait posséder plusieurs identités sur le réseau (attaque *Sybil*), permettant de contourner les méthodes de redondance pour certains protocoles. Par exemple, un capteur peut utiliser un protocole de routage assurant d'avoir toujours deux chemins disponibles pour faire transiter les données (un chemin principal et un de secours). Un capteur qui annonce plusieurs identités risque d'être choisi deux fois, rendant le protocole de routage inutile.

Un attaquant peut envoyer des paquets « hello » en utilisant une radio longue portée, ce qui fait qu'un nœud va répondre sans cesse et vider sa batterie (attaque *Hello*). Un moyen de contrer ceci est d'imposer une limite au nombre de messages envoyés par seconde ; cela n'empêche pas l'attaque mais diminue significativement la perte d'énergie. Si l'attaquant envoie des messages à une grande vitesse, il inonde le réseau et cela correspond à l'attaque par *brouillage* ou par *collisions*.

Un attaquant peut faire croire qu'un nœud est toujours en vie en envoyant des acquittements. Ceci empêche la bonne surveillance du réseau, et crée une indisponibilité.

3.2.2.4 La couche transport

L'attaque par *inondation* consiste à envoyer un grand nombre de messages à une cible afin d'épuiser ses ressources en bande passante, en mémoire ou temps de calcul. Un attaquant peut envoyer des demandes de connexion à un nœud jusqu'à ce qu'il n'ait plus de mémoire disponible, empêchant toute future tentative de connexion légitime. Une solution pour atténuer le problème est d'authentifier les messages, afin d'ignorer ceux qui ne viennent pas d'un nœud de notre réseau. Lors d'une connexion, une attaque par inondation est contrée en demandant une preuve d'écoute à notre interlocuteur avant de stocker des informations ou d'effectuer des calculs. Devoir écouter implique de ne pas inonder le réseau sur des liaisons sans fil. Nous pouvons par exemple demander une réponse à un message envoyé au client contenant une valeur aléatoire.

La dernière attaque vise la *désynchronisation* d'un appareil par un attaquant qui voudrait causer un déni de service. L'attaque consiste à envoyer un message de désynchronisation aux appareils auxquels un capteur est connecté afin qu'ils coupent la connexion.

3.2.2.5 Résumé des attaques

4	**Transport**	Inondation, Désynchronisation
3	**Réseau**	Rejeu, Trou de vers, Transfert sélectif, Puits, Sybil, Information de routage, Hello, Usurpation d'acquittement
2	**Liaison**	Collision, Épuisement, Équité
1	**Physique**	Brouillage, Dégradation

Table 3.1 – Résumé des attaques possibles dans un réseau de capteurs.

Le tableau 3.1 (extrait de [39]) résume les différentes attaques en fonction des couches réseau de la norme OSI.

L'authentification des pairs et la protection des communications permettent de lutter efficacement contre la plupart des attaques sur la couche réseau et sur les couches au-dessus, tant que les objets ne sont pas corrompus. Par exemple, l'attaque par *transfert sélectif* nécessite pour être contrée un routage à plusieurs chemins, mais si tous les messages sont protégés alors le nœud corrompu ne peut plus trier les messages. L'attaque par *rejeu* est contrée par un numéro de séquence, qui est également utilisé dans les protocoles de sécurité, par exemple TLS ou IPsec. De même, l'attaque par *inondation* est contrée par l'authentification des messages, et par un mécanisme contre le déni de service, voir 3.1.10.

Cette dernière attaque n'est pas gérée dans notre document car indépendante du fonctionnement d'une architecture, au mieux un pare-feu peut limiter les dégâts s'il détecte trop de tentatives de connexion.

Au-delà d'un objet corrompu, qui pourrait s'authentifier comme étant un nœud légitime du réseau tout en étant malicieux, le réseau peut être mis à mal par un manque de standards couvrant la protection des communications. Par exemple, la protection des communications par DTLS ne se fait qu'au-delà de la couche transport, ce qui ne permet pas de prévenir les attaques évoquées. Les standards souffrent également d'un manque de généricité, et les protocoles sont amenés à employer des moyens de protection des échanges spécifiques. Par exemple, le protocole de routage RPL décrit des messages sécurisés pour protéger les informations de routage échangées [40].

En conclusion, les attaques sur les objets contraints et sur le réseau peuvent être contrées en grande partie par l'authentification et la protection des communications, et seules les attaques de corruption des nœuds ne sont pas contrées. Cependant, malgré la sécurité offerte par la cryptographie, les standards actuels ne l'utilisent pas de manière systématique.

3.2.3 Phases de déploiement

Cette section illustre les étapes d'un déploiement et de l'exploitation d'un réseau de capteurs. Ces étapes sont expliquées plus en détail dans [41].

La première phase est l'amorçage (*bootstrapping*). Elle consiste en l'authentification d'un nœud, soit avec une identité pré-configurée dans le firmware de l'objet, soit en créant une nouvelle identité pour l'objet (aussi nommée *onboarding*, *provisioning*). Suite à cette authentification, une autorisation d'accès au réseau est donnée, qui donne lieu à la configuration de paramètres de communication (par exemple en fournissant un certificat pour des communications ultérieures). L'objet est alors enregistré et rattaché à un domaine ou à un groupe ; deux objets de ce même groupe peuvent désormais communiquer sans nécessiter un partage préalable d'informations.

La seconde phase est dite « opérationnelle ». Les objets démarrent et s'arrêtent, et répondent aux requêtes des clients. La découverte de ressources fait partie de cette phase.

Enfin, la dernière phase est la fin de vie, la mise hors service d'un objet. Cette phase implique un changement de confiance dans le réseau par rapport à l'identité de l'objet hors service. Les objets restants ne doivent plus pouvoir authentifier l'objet hors service.

3.2.4 Conclusion

Dans les environnements contraints, les algorithmes symétriques avec un secret partagé sont à privilégier par rapport aux algorithmes asymétriques, trop lents pour des matériels à faible capacité de calcul. La plupart des attaques dans ces environnements peuvent être contrées grâce à la protection des communications, mais les standards actuels ne chiffrent pas l'intégralité des communications. Par exemple, DTLS chiffre au delà de la couche transport et laisse la communication vulnérable aux attaques sur les couches de niveau réseau et transport. Enfin, notre travail se focalise principalement sur la phase opérationnelle, pour vérifier l'influence du type d'architecture lors du fonctionnement. Nous nous intéresserons également à la phase de déploiement lors de la comparaison des architectures.

3.3 Conclusion

Dans ce chapitre, nous avons introduit la sécurité des communications et les algorithmes utilisés dans les environnements contraints. Nous avons listé les attaques subies dans ces réseaux, et indiqué comment elles peuvent être contrées par l'usage de communications sécurisées. Enfin, nous avons listé les phases de déploiement d'un réseau, c'est-à-dire les phases d'amorçage, opérationnelle et de fin de vie. Ce chapitre nous a permis d'établir les bases de notre travail en matière de sécurité, et de mieux cerner le domaine d'étude.

Pour protéger leur communication, des interlocuteurs doivent prouver leur identité à leur destinataire, créer une communication sécurisée (c'est-à-dire négocier une clé et des algorithmes cryptographiques), vérifier leurs droits et enfin s'échanger des données. Dans les environnements contraints, le choix des algorithmes est limité par les capacités des matériels, et les algorithmes symétriques de protection des communication (confidentialité, intégrité et authenticité des messages) et de partage de clés sont à privilégier. Les protocoles de communication sécurisée regroupent l'ensemble des algorithmes nécessaires à l'établissement d'une communication sécurisée (authentification, partage de clés, protection des communications), mais ne sont pas tous adaptés aux environnements contraints (par exemple DTLS avec ses nombreux messages et ses en-têtes de taille importante pour les réseaux contraints comme IEEE 802.15.4).

Dans le chapitre suivant, nous fournirons une vue d'ensemble d'un déploiement en phase opérationnelle pour montrer quels sont les appareils, les interactions, les informations échangées, les protections choisies ou encore les fonctionnalités d'un réseau (cache de données, découverte de ressources, etc.), ce que nous appellerons une *architecture de sécurité*.

Chapitre 4

Architecture de sécurité

La protection des communications, ainsi que l'authentification et l'autorisation des pairs doivent êtres prises en compte dans la conception d'une solution de sécurité cohérente pour les environnements contraints. La contribution présentée dans ce chapitre est la définition d'une « architecture de sécurité », qui illustre cette solution de sécurité cohérente et délimite ce qui peut être considéré comme tel dans la littérature. De plus, cette définition nous permet de caractériser les éléments en utilisant un langage commun, c'est-à-dire de décrire les architectures et ainsi pouvoir raisonner sur les interactions, les besoins matériels ou encore les fonctionnalité des architectures. Enfin, nous verrons les forces et faiblesses de notre définition.

Pour définir une architecture de sécurité, nous sommes partis de deux hypothèses simplificatrices. Premièrement, notre travail se concentre sur les solutions de sécurité indépendantes de la couche liaison, pour laisser les opérateurs du réseau choisir le médium de communication selon la situation. Deuxièmement, nous nous concentrons sur la phase opérationnelle (voir section 3.2.3) : d'une part, cette phase illustre le fonctionnement de l'architecture lors de son utilisation par des clients et donc son comportement durant la quasi-totalité de la vie du réseau, d'autre part les architectures décrivent rarement les autres phases (amorçage et fin de vie) qui sont laissées à la discrétion des opérateurs.

Ce chapitre évoque dans un premier temps le manque de définition formelle d'une architecture de sécurité, ce qui nous amène à décrire notre première contribution, à savoir une définition formelle d'une telle architecture de sécurité, puis une méthode pour décrire les architectures trouvées dans la littérature.

4.1 Manque de définition

Nous souhaitons évaluer l'influence d'une architecture de sécurité de type maître-esclave, et la première question à laquelle nous devons répondre est : qu'est-ce qu'une architecture de sécurité ? La définition de l'organisme de standardisation ITU-T [42] (du 2004-04-15) n'est pas satisfaisante. Cette définition ne correspond pas à nos besoins, l'ITU-T décrit uniquement l'aspect sécurité et ne fournit aucune méthode de description d'une architecture, par conséquent cette définition ne permet pas de raisonner sur les architectures. La littérature manque d'une définition claire : il faut se poser la question de ce qu'est une architecture de sécurité.

Nous souhaitons une définition ainsi qu'une manière de décrire une architecture. L'objectif est de pouvoir raisonner sur les architectures, et répondre à des questions comme :

— Quelle architecture permet de séparer explicitement le service d'autorisation du service d'authentification ?
— Y a-t-il un échange d'algorithmes cryptographiques entre le client et une entité du domaine du serveur de ressources (agilité cryptographique) ?

— Sachant les informations échangées, quels sont les prérequis de l'architecture, qui doit générer quoi comme donnée ?
— Les nœuds contraints doivent-ils utiliser une horloge temps-réel pour générer une information échangée avec une autre entité ?

Une définition permet de :

1. délimiter ce qui dans la littérature peut être considéré comme une architecture de sécurité ;
2. caractériser les éléments en utilisant un langage commun ;
3. comparer les architectures notamment sur les aspects liés aux contraintes des appareils, ainsi que sur la sécurité offerte.

4.2 Composants de base d'une architecture de sécurité

Dans toute architecture de communication, nous avons deux éléments en réseau souhaitant communiquer, un client et un serveur de ressources. Tous les deux sont des logiciels, le client envoie une requête pour demander une information, et le serveur de ressources reçoit et répond à cette requête.

Dans une architecture, deux services sont nécessaires pour assurer la sécurité : le service d'authentification, pour s'assurer de l'identité des clients, et le service d'autorisation pour vérifier leurs droits.

4.3 Fonctionnalités de base

Toute architecture doit posséder un ensemble minimal de fonctionnalités pour être considérée comme sécurisée.

Premièrement, toute architecture doit effectuer une **authentification mutuelle** des interlocuteurs et **autoriser les requêtes**. Le service d'authentification doit connaître et reconnaître l'identité du client, c'est-à-dire que le client doit prouver sa possession du secret partagé ou du matériel cryptographique privé. Le service d'autorisation doit vérifier les droits du client avant que le serveur de ressources ne traite la requête. De son côté, le client doit authentifier chaque entité avec laquelle il interagit, et vérifier que chaque serveur de ressources est autorisé à effectuer les opérations demandées ; dans les deux cas, l'objectif est de prévenir une diffusion non autorisée d'une information privée ou une action interdite. Pour l'autorisation, d'une part, certains déploiements nécessitent des autorisations implicites, où une authentification du client suffit à lui conférer le droit d'accéder à toutes les ressources. Cela peut être le cas par exemple en domotique où une seule télécommande contrôle l'ensemble des objets de la maison. D'autre part, dans plusieurs cas les déploiements nécessitent une gestion plus subtile des autorisations, où la personne (ou l'organisme) en charge du domaine contraint souhaite donner le moins de droits possibles à chaque client (principe de moindre privilège). Par exemple, une entreprise de transport possède des conteneurs, chacun pouvant être localisé par GPS, et possédant une porte verrouillée et une régulation à distance de la température pour préserver les biens transportés. L'entreprise souhaite laisser le client accéder à la localisation de la marchandise, et pouvoir réguler la température via un ventilateur, mais seul un de ses employés peut ouvrir et fermer la porte.

Deuxièmement, toute architecture de sécurité doit protéger ses communications, la transmission de données doit être protégée. Chaque communication doit gérer l'**authenticité**, l'**intégrité**, et la **confidentialité des données**. Chaque interlocuteur doit pouvoir dire si un paquet reçu a été modifié (intégrité) et s'il provient du bon expéditeur (authenticité), et s'assurer que seul l'interlocuteur comprenne la transmission (confidentialité) [1]. Ensuite, toute architecture doit prévenir les attaques par le rejeu : une écoute passive ne doit pas permettre une authentification en rejouant d'anciens messages,

1. La confidentialité n'est pas nécessaire dans tous les cas, parfois l'intégrité et l'authenticité suffisent, mais c'est un cas particulier qui ne sera pas étudié ici.

ni permettre à une commande d'être interprétée plusieurs fois par exemple. Toute autre protection est considérée optionnelle, car répond à des besoins précis qui sortent du cadre général. Par exemple, la protection contre le déni de service est optionnelle dans le cadre de communications locales, ou encore la vérification de l'origine des données est dispensable si les serveurs de ressources sont cachés au client, comme il est possible de le faire dans une architecture de type maître-esclave.

Pour conclure, une solution de sécurité cohérente doit :

— vérifier l'identité de ses clients et leurs droits (authentification et autorisation) ;

— fournir des contre-mesures à une écoute passive ou active, c'est-à-dire contrer des attaques liées à l'altération et au rejeu de messages, ou à la création de paquets malicieux.

La solution de sécurité doit donc chiffrer, authentifier et vérifier l'intégrité des messages, authentifier les clients et autoriser les requêtes, et gérer le matériel cryptographique. Toute solution respectant ces conditions est considérée comme une architecture de sécurité.

4.4 Définition formelle

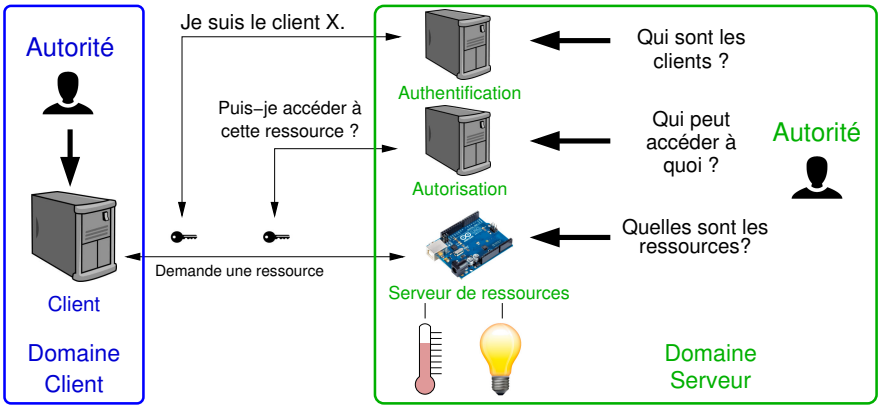


FIGURE 4.1 – Exemple de composants dans une architecture de sécurité.

Nous définissons une architecture de sécurité comme un quadruplet $<E,\ A,\ D,\ R>$:

1. les entités (E) sont les composants logiciels fournissant les fonctionnalités dans une architecture ;
2. elles sont contrôlées par des autorités (A) qui sont des personnes ou des organisations ;
3. les domaines de sécurité (D) représentent le lien entre les autorités et les ensembles d'entités qu'elles contrôlent ;
4. les relations (R) sont les interactions entre les entités.

Le schéma 4.1 illustre ces concepts. Un client demande une ressource, mais doit être authentifié (« est-ce que vous me reconnaissez ? »), et être autorisé (« est-ce que je peux effectuer cette action ? »). Pour délimiter les domaines sur le schéma, l'autorité serveur (c'est-à-dire l'autorité contrôlant le serveur de ressources, le service d'authentification et celui d'autorisation) et ses entités sont de couleur verte. L'autorité cliente (c'est-à-dire l'autorité contrôlant le client) effectuant la requête et son entité cliente sont en couleur bleue.

4.4.1 Entités (E)

Les entités représentent le logiciel déployé, sur un appareil, fournissant un service dans l'architecture. Nous avons 4 entités de base dans une architecture :

1. le client demandant une ressource ;
2. le serveur de ressources fournissant une représentation d'une ressource provenant des capteurs ou activant des actionneurs ;

3. le service d'authentification du client vérifiant que seul un client connu peut accéder au domaine du serveur ;
4. le service d'autorisation donnant le droit d'accéder à une ressource à un ensemble de clients.

De multiples entités peuvent fonctionner sur un même appareil.

4.4.2 Autorités (A)

Les autorités sont les personnes ou les organisations en charge des entités, elles contrôlent leur comportement et leur politique. Par exemple, l'autorité serveur est en charge des entités d'authentification et d'autorisation pour accéder au serveur de ressources.

4.4.3 Domaines de sécurité (D)

Les domaines de sécurité représentent un ensemble d'entités contrôlées par une autorité. Chaque domaine a ses propres mécanismes, politiques de sécurité, gestion de clés et ses interactions entre ses entités. Un domaine contrôlé par une autorité a est défini comme $Domaine(a) = \{e \in E \mid a \text{ contrôle } e\}$. Par conséquent, D est défini comme :

$$D = \{Domaine(a) | a \in A\}$$

Excepté quand une architecture définit explicitement ses domaines, nous considérerons que l'autorité du domaine du serveur contrôle toutes les entités sauf le client. Par exemple, une entreprise permet à des ordinateurs (clients) d'une autre organisation (autre domaine) d'utiliser ses imprimantes (serveurs de ressources), une fois authentifiés sur un serveur de l'entreprise (entités d'authentification et d'autorisation).

4.4.4 Relations (R)

Les relations décrivent les interactions entre les entités dans une architecture. Une relation est un quadruplet $< e_1 \text{ op } e_2,\ p,\ o,\ i >$
où $e_1 \in E, e_2 \in E, \text{op} \in \{\rightarrow, \leftrightarrow\}, p \subseteq P, o \subseteq O, i \subseteq I$.

1. E est un ensemble d'entités détaillé précédemment ;
2. op représente la direction d'information entre deux entités ;
3. l'opération $\leftrightarrow$ est utilisée pour indiquer un échange bidirectionnel de messages pour simplifier les relations pour des raisons de lisibilité ;

   Par exemple, pour grouper les messages d'une connexion DTLS.
4. P est l'ensemble des protocoles utilisés dans l'architecture, chaque relation possède un sous-ensemble p de ces protocoles. ;
5. O est l'ensemble des propriétés utilisées, chaque relation possède un sous-ensemble o de ces propriétés.

   Il existe un grand nombre de propriétés de communication, ce document se concentre sur l'ensemble suivant pour décrire les propriétés inhérentes des protocoles des relations :
   — fiabilité F ;
   — ordre O ;
   — confidentialité C, authentification mutuelle MA, intégrité I ;
   — disponibilité A (protection contre le déni de service) ;
   — protection contre le rejeu R ;
   — responsabilité AC.
6. I est l'ensemble des informations échangées dans l'architecture, le sous-ensemble i représente les informations échangées dans une relation, par exemple pour :

— authentifier mutuellement les deux parties d'une communication ;
— négocier les algorithmes ;
— effectuer une requête.

R est donc l'ensemble des relations qui caractérisent l'implémentation d'une architecture.

Une demande d'une ressource à un serveur de ressources (SR) peut être représentée par les relations telles que dans le tableau suivant.

	entités	protocoles	propriétés	informations
R_1	C → SR	CoAP	F	URI
R_2	SR → C	CoAP	F	valeur

FIGURE 4.2 – Exemple de relations.

Par exemple, la figure 4.2 montre deux relations, qui décrivent une simple requête émise par un client C vers un serveur de ressources SR, et la réponse, en utilisant seulement le seul protocole CoAP. C envoie un message à SR en s'assurant que le message soit bien reçu (fiabilité F), et ce message contient une URI, puis SR répond en donnant la valeur attendue. Pour simplifier la lecture, les protocoles de couches inférieures à la couche transport sont ignorés ; seules leurs propriétés sont prises en compte.

	entités	protocoles	propriétés	informations
R_1	C → SR	CoAP	F, C, I, MA	URI
R_2	SR → C	CoAP	F, C, I, MA	valeur

FIGURE 4.3 – Exemple de relations, représentant une requête protégée.

La figure 4.3 montre deux relations qui décrivent une requête protégée entre C et SR. C envoie un message à SR, en utilisant le protocole CoAP, en s'assurant que le message soit bien reçu (fiabilité F) et en protégeant la communication. La communication est protégée grâce aux propriétés de chiffrement, d'intégrité et d'authentification. Dans cet exemple, une session est déjà établie, la requête passe par un canal de communication existant, et ce message contient une URI. SR répond au client, la réponse est également protégée.

	entités	protocoles	propriétés	informations
R_1	C → SR	CoAP		infoa, infob, $[\ \text{URI}\ ]_{k_{c\to SR}}$
R_2	SR → C	CoAP		infox, infoy, $[\ \text{valeur}\ ]_{k_{SR\to c}}$

FIGURE 4.4 – Exemple de relations, représentant une requête protégée.

La figure 4.4 montre deux relations, qui décrivent une requête protégée. Deux entités discutent, C et SR. C envoie un message à SR, utilisant le protocole CoAP, en protégeant l'URI de la requête, représenté par $[\ \text{URI}\ ]_{k_{c\to SR}}$. Cette notation couvre le chiffrement, l'intégrité et l'authentification de la donnée. L'information *URI* est protégée par une clé symétrique (k), utilisée pour protéger des informations de C à SR. Nous utilisons cette représentation dans les relations lorsqu'une couche inférieure ne chiffre pas entièrement la couche applicative, sinon nous utilisons la propriété C. Les autres informations échangées dans ce message ne sont pas protégées. SR répond, la valeur est également protégée et la notation $[\ \text{valeur}\ ]_{k_{SR\to c}}$ indique que `valeur` est protégée par une clé symétrique (k), utilisée pour protéger des informations de SR à C. La notation introduite permet de protéger seulement une partie des informations échangées, ce qui est nécessaire pour certaines architectures.

La figure 4.5 montre deux relations, qui décrivent un échange. La notation $\perp$ représente un ensemble de valeurs inconnues dans les relations (protocoles, propriétés, informations) ; plus tard ces relations seront indiquées dans les schémas des architectures par des liens bleus. La notation $\leftrightarrow$ décrit une

	entités	protocoles	propriétés	informations
R_1	C $\leftrightarrow$ SR	CoAP	$\perp$	infoa, infob, requête
R_2	SR $\rightarrow$ C	CoAP	$\perp$	réponse

FIGURE 4.5 – Exemple de relations abrégées, avec des propriétés inconnues.

série abrégée de relations, pour des raisons de lisibilité. Par exemple, le protocole DTLS étant connu, ses relations sont abrégées dans la description d'une architecture pour aller à l'essentiel, c'est-à-dire montrer les particularités de cette architecture, plus tard ces relations seront indiquées dans les schémas des architectures par des liens magenta.

	entités	protocoles	propriétés	informations
R_1	C $\rightarrow$ SR	CoAP	$\perp$	infoa, infob, requête
R_2	SR $\rightarrow$ AZ	$\perp$	$\perp$	$\perp$
R_3	AZ $\rightarrow$ SR	$\perp$	$\perp$	autorisation
R_4	SR $\rightarrow$ C	CoAP	$\perp$	réponse

FIGURE 4.6 – Exemple de relations montrant deux entités non distinctes.

La figure 4.6 montre quatre relations. Elles décrivent une requête de C vers SR, ensuite autorisée par une entité AZ, mais nous ne connaissons pas les informations échangées ni comment. Ce manque d'informations sur les échanges marque deux entités qui ne sont pas distinguées dans l'architecture par des communications explicites entre ces deux entités, ce qui impliquerait de pouvoir séparer les entités sur des matériels différents. Nous souhaitons mettre en avant les différentes entités dans nos figures pour représenter le déroulement des opérations nécessaires dans une architecture de sécurité ; il est ainsi possible de voir les opérations internes aux architectures et leur ordre.

À travers ces quelques exemples, nous venons de voir comment décrire des relations. Une relation peut représenter un échange sécurisé par un protocole, les propriétés de la relation sont alors indiquées, avec au minimum : chiffrement, authenticité et intégrité. L'échange sécurisé peut aussi se faire en ne protégeant qu'une partie des données échangées, notre description prend alors en compte ceci en précisant les données protégées et avec quelles clés, comme $[\text{ donnée }]_{k_C}$. Enfin, si un échange n'est pas décrit dans un protocole, par exemple pour une demande d'autorisation (implicite dans le protocole), alors les propriétés et protocoles sont indiqués comme inconnus via la notation $\perp$.

4.4.5 Conclusion

Une architecture de sécurité est définie par au moins quatre entités, qui représentent des fonctionnalités nécessaires : un client, un serveur de ressources pour fournir les données et recevoir des actions à effectuer, un service d'authentification et un autre d'autorisation pour vérifier respectivement l'identité et les droits des clients. Des domaines regroupent les entités se faisant confiance ; par exemple le domaine serveur regroupe les entités d'authentification, d'autorisation et de serveur de ressources. Chaque domaine est géré par une autorité qui donne les secrets partagés aux entités pour qu'elles se fassent confiance. Enfin, les interactions entre les entités sont décrites dans des relations ; une relation indique qui est l'émetteur et le récepteur, les protocoles utilisés, les propriétés (chiffrement, authenticité, etc.), et enfin la donnée échangée.

4.5 Propriétés minimales de sécurité

Pour être considérée comme sûre, une architecture doit posséder un ensemble minimal de propriétés. Ces propriétés protègent les communications, ainsi que le comportement global de l'architecture.

4.5.1 Protection des données

Une architecture de sécurité doit fournir une protection des données durant le transfert entre deux entités, l'architecture doit :

— vérifier la confidentialité C ;
— vérifier l'authentification MA ;
— vérifier l'intégrité I ;
— protéger contre le rejeu R.

D'autres propriétés optionnelles peuvent caractériser des communications, telles que par exemple :

— la protection de disponibilité A, pour prévenir des attaques de surcharge de mémoire avec des connexions à moitié ouvertes par exemple ;
— la vérification de l'origine des données, pour connaître le serveur de ressources qui a produit la donnée ;
— la dissimulation de taille de paquet ;
— la création de faux paquets, pour cacher le vrai usage du réseau.

Chaque relation a un ensemble de ces propriétés, dans certaines architectures, seule une partie des données est protégée. Aussi, des propriétés sans lien avec la sécurité telles que la fiabilité F et l'ordre O peuvent être requises ou induites par la couche de sécurité. Pour assurer la protection des données, une architecture peut créer un canal de communication ou utiliser des objets de sécurité (voir chapitre 3.1.8).

4.5.2 Authentification mutuelle

Une authentification mutuelle entre le client et l'entité d'authentification doit être effectuée (voir chapitre 3.1.3), le client est alors authentifié au domaine de sécurité. Les mécanismes d'authentification sont dépendants de la situation. Dans un réseau local, avec des appareils contrôlés par une seule entité, utiliser des clés pré-configurées comme mécanisme d'authentification est un scénario réaliste. Si le déploiement comporte de nombreux clients, les certificats avec une durée de validité limitée peut être un meilleur choix : le serveur de ressources effectuant l'authentification n'aura pas à garder un état pour chaque client. Enfin, si les certificats ont une durée de validité limitée, le serveur de ressources doit vérifier la date d'expiration des certificats (une horloge est alors requise) et la signature (une clé publique et par conséquent la cryptographie asymétrique sont nécessaires).

4.5.3 Autorisation

L'architecture peut utiliser les autorisations pour informer le client de ses droits (quelle opération effectuer sur quel serveur de ressources). Par exemple, ces règles vont discriminer un client ou une connexion sur certains éléments disponibles, comme par exemple l'identité du client ou la date de connexion. Puis, le client aura accès à un ensemble de ressources, serveurs de ressources ou encore un type de ressources sur différents serveurs.

4.5.4 Gestion des clés

La gestion des clés doit être prévue dans l'architecture de sécurité, pour assurer la protection des communications, voir section 3.1.5 (page 19).

4.5.5 Autres mécanismes

D'autres mécanismes peuvent être employés pour faciliter le déploiement ou ajouter des fonctionnalités, par exemple :

— l'*agilité cryptographique* [34] permet de choisir les algorithmes cryptographiques à la connexion et éviter de changer un protocole à chaque mise à jour des algorithmes ;

— la *découverte de ressources* pour explorer les ressources disponibles ou pour gérer des ressources dynamiques ;
— un *service d'historique*, gardant en mémoire les requêtes effectuées par les clients pour une analyse ultérieure.

4.5.6 Conclusion

Les propriétés minimales dans une architecture de sécurité sont la confidentialité, l'intégrité et l'authenticité des données, ainsi que la protection contre le rejeu. À cela s'ajoute l'authentification mutuelle des interlocuteurs et l'autorisation des requêtes. Enfin, l'architecture de sécurité doit prévoir le partage de clés.

4.6 Exemple d'architecture de sécurité

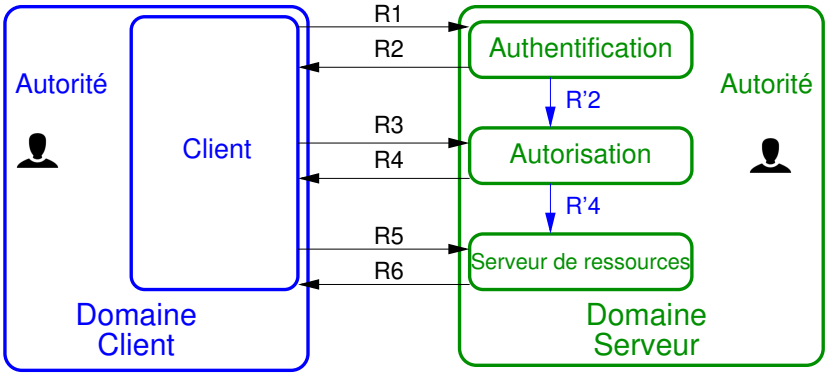


FIGURE 4.7 – Exemple d'architecture de sécurité.

La figure 4.7 représente un exemple d'architecture de sécurité selon notre définition. Deux autorités sont décrites, l'autorité *cliente* et *serveur*. L'architecture présente un ensemble minimal d'entités : un client, un serveur de ressources, et des entités d'authentification et d'autorisation. Les domaines sont ceux du client (comportant a minima l'entité *client*) et du serveur (comportant le serveur de ressources et les entités d'authentification et d'autorisation) Plusieurs clients (de différents domaines) peuvent demander des ressources au domaine serveur. Deux domaines sont représentés sur la figure :
$Domaine(client) = \{Client\}$
$Domaine(serveur) = \{Authentification, Autorisation, ServeurdeRessources\}$

	entités	protocoles	propriétés	informations
R_1	C → AN	CoAP	F	id client
R_2	AN → C	CoAP	F	$[\text{ ticket }]_{k_C}$ ticket = temps t, signature AN
R'_2	AN → AZ	⊥	⊥	⊥
R_3	C → AZ	CoAP	F	$[\text{ ticket, requête }]_{k_C}$
R_4	AZ → C	CoAP	F	$[\text{ ticket', requête }]_{k_C}$ ticket' = temps t, signature AZ
R'_4	AZ → SR	⊥	⊥	⊥
R_5	C → SR	CoAP	F	$[\text{ ticket', requête }]_{k_C}$
R_6	SR → C	CoAP	F	$[\text{ réponse }]_{k_C}$

FIGURE 4.8 – Relations de notre architecture exemple.

La figure 4.8 possède 4 entités, avec des noms abrégés : client (C), service d'authentification (AN), service d'autorisation (AZ), serveur de ressources (SR) [2]. Seul le protocole CoAP est utilisé.

Dans cette architecture, nous voyons des relations qui se passent en parallèle (R_2 et R'_2, R_4 et R'_4). Les relations avec le client sont :

- R_1. le client envoie son identité au service d'authentification, CoAP est utilisé dans cet échange et la communication est fiable (par CoAP ou une autre couche) ;
- R_2. le service d'authentification lui donne un ticket, protégé via une clé partagée connue de C ;
- R'_2. cette relation est non spécifiée, aucune supposition n'est faite, $\perp$ représente un ensemble inconnu ;
- R_3. le client envoie le ticket et une requête chiffrée par la clé partagée k_C au service d'autorisation. La requête est une ressource et une URI action ;
- R_4. le service d'autorisation valide la requête, et envoie au client un nouveau ticket (ticket') qu'il a signé ;
- R'_4. comme pour la relation R'_2, aucune supposition n'est faite sur le contenu de cet échange ;
- R_5. le client envoie ce ticket au serveur de ressources ;
  Le ticket est une preuve de confiance et d'autorisation pour effectuer la requête.
- R_6. Enfin, le serveur de ressources répond au client.

Deux relations sont présentées sans que nous ayons d'informations sur ce qui est échangé, ni comment (indiqué dans les figures par des flèches bleues) : R'_2 et R'_4. R'_2 illustre un message envoyé du service d'authentification au service d'autorisation, R'_4 illustre un message envoyé du service d'autorisation au serveur de ressources. Ces relations peuvent représenter, dans cet exemple, le service d'authentification partageant un secret avec le service d'autorisation (relation R'_2), puis avec le serveur de ressources (relation R'_4). Des relations non spécifiées correspondent à des communications internes sur un appareil ou hors du cadre des communications étudiées (ou non documentées par exemple).

L'architecture exemple requiert seulement la propriété de fiabilité (F), ce qui peut être atteint à la couche applicative avec CoAP (avec des messages qui demandent une réponse) ou requis à la couche liaison. Cette architecture n'a pas de propriétés de sécurité, donc aucun tunnel sécurisé n'est présent dans une couche inférieure (avec un protocole de sécurité comme DTLS par exemple). Ce manque de protocole de sécurité implique que la cryptographie soit réalisée au niveau de la couche applicative, et soit présenté dans les relations de la manière suivante : $[\text{ donnée en clair }]_{clé}$, et seule la clé k_C est utilisée.

Les informations partagées sont $I = \{\text{client id, ticket, ticket', requête, réponse}\}$. L'identité du client est une chaîne arbitraire, le contenu des tickets est détaillé dans les relations, et la requête et sa réponse sont liées à CoAP. Une clé k_C est utilisée par différentes entités, sans messages pour s'accorder dessus ce qui indique une clé pré-configurée et une authentification basée sur cette clé. Les deux tickets indiquent que, dans cet exemple, le mécanisme d'autorisation est *explicite* et les messages sont échangés pour accorder une permission. Les tickets sont protégés par une clé de chiffrement symétrique, et contiennent deux estampilles temporelles, l'une fournie par le service d'authentification, l'autre fournie par le service d'autorisation, puis utilisée finalement par le serveur de ressources. Par conséquent, la présence de ces estampilles implique que les interlocuteurs doivent savoir gérer des temps (c'est-à-dire avoir du matériel spécialisé comme une horloge temps-réel). Le ticket fourni par le serveur d'autorisation contient en plus une signature, le serveur d'autorisation doit par conséquent posséder du matériel spécialisé ou une capacité de calcul suffisante pour créer des signatures numériques. La signature est ensuite envoyée au serveur de ressources, par conséquent le serveur de ressources doit lui aussi posséder du matériel spécialisé ou une capacité de calcul suffisante pour vérifier des signatures numériques.

2. Dans le cas où des entités d'authentification et d'autorisation sont également présentes dans le domaine du client, les préfixes « C » et « S » sont utilisés tels que *CAN* et *CAZ* pour les entités d'authentification et d'autorisation du client (respectivement *SAN* et *SAZ* pour le domaine du serveur).

Dans les architectures présentées au chapitre suivant, l'autorité client et l'autorité serveur sont omises des figures puisqu'elles gèrent toujours leurs domaines respectifs.

4.7 Forces et faiblesses de cette définition

Notre travail comble un manque de définition claire, permettant de raisonner sur les architectures existantes, de les décrire et de les comparer. Par exemple, notre définition permet de décrire une architecture et d'en tirer une approximation du nombre de messages échangés entre les appareils et ainsi connaître le nombre de RTT [3] lors d'une requête. De même, la description d'une architecture avec notre définition permet de déterminer les besoins matériels des différents appareils, comme la nécessité de posséder une horloge temps-réel, les capacités de calcul ou de mémoire requises, etc. Notre définition permet également d'avoir une vue d'ensemble pour mieux étudier et ainsi comprendre les mécanismes d'authentification et d'autorisation, et visualiser les échanges, y compris internes aux appareils.

Notre définition n'est pas exempte de défauts, qui résultent de choix pragmatiques pour ne pas alourdir la description. Premièrement, il n'est pas possible d'indiquer des propriétés ne s'appliquant qu'à une partie des informations échangées. Si une architecture implique une protection partielle des communications, c'est-à-dire sans protocole sécurisé chiffrant toutes les informations échangées, les propriétés de chiffrement, d'authenticité et d'intégrité ne peuvent être indiquées, car elles s'appliquent à toutes les informations échangées. Dans ce cas, il faut donc indiquer le chiffrement (ou la signature) uniquement des données concernées, et cela entraîne une certaine lourdeur, mais qui ne peut pas forcément être abstraite sans perte d'information. Deuxièmement, les informations échangées ne sont pas classées en catégories pour ne pas compliquer la définition ni alourdir la notation et la description de l'architecture. Un classement des informations permettrait d'abstraire des détails des protocoles, et simplifier les raisonnements sur les échanges. Troisièmement, nous avons fait le choix de ne pas ajouter des entités d'authentification et d'autorisation dans le domaine client de manière systématique. Le client effectue par lui-même les actions d'authentification des entités avec lesquelles il interagit, et il vérifie les autorisations des différents serveurs de ressources sans que cela ne soit décrit par l'intermédiaire d'une entité. Quatrièmement, nous verrons au chapitre suivant des architectures qui ne prévoient qu'un seul matériel effectuant les opérations d'authentification, d'autorisation et de serveur de ressources. Les interactions s'effectuent en premier lieu entre un client et le serveur de ressources, qui transmet ensuite les messages aux entités d'authentification ou d'autorisation. L'ordre de ces interactions pourrait être repensé pour éviter l'interaction entre l'entité client et serveur de ressources avant l'authentification du client. Enfin, notre description n'est pas suffisamment précise pour être exécutée par une machine de Turing : c'est un compromis entre la formalisation (pour définir les termes et les concepts) et la lisibilité pour un humain. Une description plus automatique, plus formelle pour être exécutable pourra faire l'objet de travaux futurs.

4.8 Conclusion

Nous avons défini dans ce chapitre le terme *architecture de sécurité*. Une architecture de sécurité est un ensemble d'entités, qui sont des composants logiciels fournissant les fonctionnalités. Les entités nécessaires dans une architecture de sécurité sont :

1. client, qui demande des ressources ou envoie des commandes ;
2. serveur de ressources, qui fournit des données ou donne accès à des actionneurs ;
3. entité d'authentification des pairs ;
4. entité d'autorisation des requêtes.

3. RTT : Round-Trip Time, le temps d'un aller-retour.

Les entités sont contrôlées par des autorités, qui sont des personnes physiques ou des organisations, au sein de domaines de sécurité. La répartition en domaines permet d'exprimer une politique commune et indique la confiance entre les entités. Les interactions entre les entités sont décrites dans des « relations ». Enfin, une liste de propriétés minimales a été définie afin d'assurer la protection des communications : chiffrement, authenticité, intégrité, rejeu.

Notre définition montre qu'une architecture de sécurité couvre :

— la protection des communications, avec l'authenticité, l'intégrité et la confidentialité des messages ;
— la vérification de l'identité des pairs ;
— le contrôle d'accès ;
— la gestion du matériel cryptographique.

Le chapitre suivant utilise notre définition, ses termes et ses concepts, pour décrire puis comparer des architectures trouvées dans la littérature.

Chapitre 5

Application de la définition d'une architecture de sécurité

Dans le chapitre précédent nous avons défini une solution de sécurité cohérente désignée par « architecture de sécurité ». Nous souhaitons maintenant comparer les architectures. Pour cela il faut savoir quelles sont les architectures existantes, et avoir un résumé de chaque architecture [1] : les objectifs, le fonctionnement (entités impliquées, interactions...), les besoins matériels, ainsi que les avantages et les inconvénients. Nous comparons ensuite ces architectures sur plusieurs aspects relatifs aux contraintes des environnements (nombre d'interactions requises, taille des paquets, informations stockées et échangées, méthodes d'authentification, algorithmes requis...). Cette comparaison s'inscrit dans un contexte d'environnement contraint, et suit des scénarii montrant des prérequis différents (dans l'authentification ou la gestion des clés). Certaines architectures ne sont pas adaptées à ces environnements et notre analyse permettra de le montrer.

Ce chapitre présente les contributions suivantes :

1. La description des architectures de sécurité trouvées dans la littérature ;
2. Une comparaison de ces architectures.

5.1 Mécanismes non retenus

Dans cette section, nous nous posons la question des mécanismes qui doivent être examinés. Les mécanismes considérés dans ce document correspondent à notre définition et proviennent de la littérature académique ou de standards de fait (via principalement les documents de l'IETF, mais aussi d'OASIS). Nous n'établissons pas une liste exhaustive des architectures de sécurité, et certaines sont mises de côté. Nous listons ainsi les mécanismes qui ne seront pas étudiés dans le reste de ce document car ils ne sont pas adaptés à nos besoins (avec l'utilisation de cryptographie asymétrique) ou leur fonctionnement est identique à d'autres mécanismes étudiés.

5.1.1 Les mécanismes Extended Authentication Protocol (EAP)

EAP [43] est une méthode pour négocier un mécanisme d'authentification. Des dizaines de mécanismes d'authentification EAP sont disponibles et nous n'allons pas les étudier. Certains mécanismes d'authentification sont déjà étudiés (comme par exemple IKE), et EAP ne leur apporte qu'un profil d'usage et quelques messages de négociation supplémentaires. Certains mécanismes sont spécifiques à des situations, comme l'authentification via l'usage d'un module SIM (EAP-SIM [44]) ou dans les

1. Quand cela est approprié, un déploiement le plus contraint possible avec l'architecture est également présenté, lorsque les entités sont déployées sur des appareils contraints séparés.

réseaux téléphoniques en général (comme par exemple EAP-AKA [45] ou EAP-AKA' [46]) ou encore à l'usage de matériel spécialisé (EAP-POTP [47]). D'autres mécanismes utilisent des algorithmes asymétriques, les rendant incompatibles avec les réseaux contraints, comme PEAP [48], EAP-TTLS [49], EAP-EKE [50] etc. De plus, certains autres mécanismes ont des problèmes de propriété intellectuelle. Enfin, nous pouvons citer le mécanismes EAP-PSK [51] qui permet une authentification en peu de messages, mais sans protection contre le déni de service ni agilité cryptographique contrairement à DTLS.

Nous n'étudierons pas les mécanismes EAP car ils sont soit incompatibles avec nos objectifs (utilisation obligatoire de cryptographie asymétrique), soit déjà étudiés par ailleurs avec d'autres mécanismes qui ont des fonctionnement similaires, comme le mécanisme DTLS. Le nombre de messages change, les informations échangées changent, mais le fonctionnement est identique : négocier des algorithmes à utiliser, puis démarrer une communication par session entre deux objets.

5.1.2 Les mécanismes Authentication Authorization Accounting (AAA)

Les mécanismes AAA sont utilisés pour séparer les différentes entités sur des serveurs différents (Authentification, Autorisation), ce qui sera étudié avec le mécanisme Kerberos. Ils sont utilisés principalement pour le contrôle d'accès d'appareils en réseau. Ces mécanismes sont :

- Remote Authentication Dial-In User Service (RADIUS) [52] et [53], un mécanisme utilisé chez les fournisseurs d'accès à Internet pour authentifier les clients et faire du suivi de consommation. RADIUS est également utilisé sur les réseaux cellulaires (LTE, UMTS), pour autoriser les mobiles à accéder au réseau.
- DIAMETER [54], successeur de RADIUS, corrigeant certaines limitations.
- Terminal Access Controller Access-Control System Plus (TACACS+) [55], un mécanisme qui fait la distinction entre les mécanismes d'authentification, d'autorisation et de traçabilité. Le chiffrement est de bout-en-bout (pas seulement du mot de passe comme dans RADIUS).

Nous n'étudierons pas les mécanismes AAA car ils sont déjà étudiés par ailleurs avec le mécanisme Kerberos dont le fonctionnement est similaire.

5.1.3 Les mécanismes de single sign on (SSO)

Leur objectif est de ne nécessiter qu'une seule authentification pour accéder à des ressources différentes (par exemple, des sites web différents). Nous pouvons citer quelques mécanismes de SSO répandus, comme CAS, OAuth ou encore OpenID Connect qui permet une autorisation fine des droits, accordés par l'utilisateur à chaque ressource (site web), et ce mécanisme est de plus en plus intéressant du fait du règlement n° 2016/679, dit règlement général sur la protection des données (RGPD).

Ces mécanismes ne seront pas étudiés car ils sont liés au contexte particulier de l'authentification des applications web.

5.1.4 Conclusion

Nous n'étudierons pas les mécanismes EAP et AAA car ils n'apportent pas une différence substantielle de fonctionnement avec les architectures étudiées. Le mécanisme SSO n'est pas étudié car il ne correspond pas au contexte de notre document.

En revanche, des mécanismes de base existent pour établir des communications sécurisées (DTLS, IPsec, etc.) et les déploiements peuvent se contenter de les utiliser. Cela représente des architectures utilisées en pratique même si elles ne sont pas décrites en tant que telles dans la littérature. Nous allons décrire ces architectures formellement, pour pouvoir les comparer à des architectures plus complexes.

5.2 Étude des architectures

Dans cette section, nous présentons l'ensemble des architectures étudiées et nous les formalisons selon notre définition.

5.2.1 ACE DCAF

ACE DCAF (Authentication and Authorization for Constrained Environments Delegated CoAP Authorization Framework) est une architecture décrite dans un *draft* IETF [56] et des usages possibles sont présentés dans [57].

L'objectif de ACE DCAF est de fournir une architecture permettant une autorisation fine dans les environnements contraints. La communication entre le client et le serveur de ressources est autorisée des deux côtés, dans le domaine de sécurité du client et du serveur, le client ne peut donc seulement échanger des informations qu'avec le serveur de ressources si les deux autorités (celle du domaine client et celle du domaine serveur) sont d'accord.

Pour atteindre cet objectif, chaque domaine possède ses propres entités d'authentification et d'autorisation :

— dans le domaine client, ces entités limitent la durée de communication entre le client et les serveurs de ressources en fournissant un matériel cryptographique avec une faible durée de validité. Ces entités empêchent un client malicieux de demander indéfiniment des ressources ;
— dans le domaine serveur, ces entités contrôlent de manière centralisée les connexions et le matériel cryptographique ; elles sont déployées sur un appareil plus puissant que les serveurs de ressources lorsque le déploiement s'y prête.

Par conséquent, chaque domaine contrôle précisément les communications de chacune de ses entités avec l'autre domaine

L'architecture est suffisamment générique pour être pertinente dans plusieurs cas. D'une part, elle est pertinente lorsque les entités d'un des domaines sont sur un seul appareil, permettant le déploiement à des endroits où un appareil supplémentaire pour l'authentification et le contrôle d'accès n'est pas envisageable ou est superflu. D'autre part, elle est également pertinente lorsque les domaines ont des entités sur plusieurs appareils, ce qui permet de centraliser les authentifications et les autorisations.

5.2.1.1 Entités, Autorités, Domaines, Relations

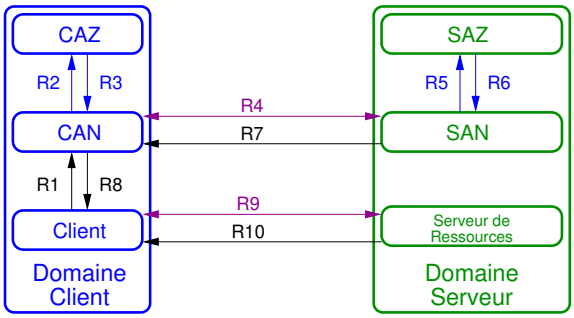


FIGURE 5.1 – Modèle de l'architecture ACE DCAF.

La figure 5.1 représente un modèle de l'architecture ACE DCAF. Deux domaines sont définis :

— $Domaine(client) = \{\text{CAN, CAZ, Client}\}$
— $Domaine(serveur) = \{\text{SAN, SAZ, Serveur de Ressources}\}$

Chaque domaine a des entités d'authentification et d'autorisation. Dans chaque domaine, les entités d'authentification et d'autorisation sont liées [2].

2. Les relations entre ces entités ne sont pas spécifiées, elles sont supposées être sur le même appareil (voir section 4.4.4).

	entités	protocoles	propriétés	informations
R_1	C → CAN	DTLS, CoAP	C, I, MA, R	requête = { URI AS, URI ressource, action, temps t }
R_2	CAN → CAZ	⊥	⊥	⊥
R_3	CAZ → CAN	⊥	⊥	autorisation
R_4	CAN ↔ SAN	DTLS ∨ TLS, CoAP	F, O, C, I, MA, A, R, AC	id_{CAN}, preuve id_{CAN}, id_{SAN}, preuve id_{SAN}, requête preuve id ∈ { certificat, PSK, clé publique brute } *connexion DTLS abrégée (lisibilité)*
R_5	SAN → SAZ	⊥	⊥	⊥
R_6	SAZ → SAN	⊥	⊥	autorisation
R_7	SAN → CAN	DTLS, CoAP	C, I, MA, R	ticket = { info autorisation serveur, $SAN_{temps\ t}$ $SAN_{durée\ de\ validité}$, clé de session }
R_8	CAN → C	DTLS, CoAP	C, I, MA, R	ticket, CAN_{info} CAN_{info}= { info autorisation client, $CAN_{temps\ t}$, $CAN_{durée\ de\ validité}$}
R_9	C ↔ SR	DTLS, CoAP	F, O, C, I, MA, A, R, AC	requête, id, preuve id id : {info autorisation serveur, $SAN_{temps\ t}$, $SAN_{durée\ de\ validité}$} preuve id : { clé de session } *connexion DTLS puis requête, abrégées (lisibilité)*
R_{10}	SR → C	DTLS, CoAP	C, I, MA, R	réponse *connexion établie, SR répond aux requêtes jusqu'à la déconnexion de C ou expiration de la connexion*

Table 5.1 – Relations de l'architecture ACE DCAF.

La table 5.1 et la figure 5.1 décrivent les relations dans ACE DCAF. Nous considérons que :

1. les entités sont sur des appareils physiques distincts (excepté pour les entités d'authentification et d'autorisation), comme présenté dans le *draft* IETF ;
2. le client et le serveur de ressources sont déployés sur des appareils contraints, la communication est alors entre quatre appareils (C, SR, CAN+CAZ, SAN+SAZ) ;
3. une session est déjà établie entre :
   — le client et son service d'authentification (C et CAN) ;
   — le serveur de ressources et son service d'authentification (SR et SAN).

Les relations correspondent à :

1. (R_1) le client contacte ses entités d'authentification et d'autorisation, pour savoir s'il peut et comment contacter le serveur de ressources ;
2. (R_2 et R_3) les entités d'authentification et d'autorisation du domaine client se concertent pour autoriser ou non l'accès à la ressource ;
3. (R_4) elles contactent leurs équivalents dans le domaine serveur ;
4. (R_5 à R_8) le droit d'accès à la ressource est vérifié, puis un ticket est envoyé au client en passant par l'entité d'authentification du client ;
   Le ticket inclut le matériel cryptographique et une représentation d'autorisation.
5. (R_9 et R_{10}) enfin, le client contacte le serveur de ressources, en créant une session DTLS, puis en envoyant une requête, à laquelle le serveur de ressources répond.
   La session DTLS utilise des clés pré-configurées, fournies dans le ticket. La connexion finale peut être persistante (dans le cas de demandes multiples).

5.2.1.2 Avantages et inconvénients

Les avantages de cette architecture sont :

— la flexibilité : l'architecture permet l'usage d'un appareil dédié avec les entités d'authentification et d'autorisation, dans un seul ou les deux domaines. Par conséquent, des procédures d'authentification et d'autorisation complexes peuvent être envisagées sur des appareils appropriés, dans chaque domaine. Par exemple, une authentification avec certificats et des algorithmes cryptographiques lents.

— un seul matériel cryptographique entre le serveur de ressources et le client : ce matériel cryptographique est utilisé pour contacter leurs entités d'authentification et d'autorisation respectives. Un nouveau matériel cryptographique de court terme est créé à chaque nouvelle connexion, et détruit immédiatement après. Par conséquent, un client peut contacter des dizaines de serveurs de ressources sans nécessiter beaucoup de mémoire, et un serveur de ressources peut avoir des dizaines de clients Un compromis est à faire entre les durées.
— Enfin, chaque domaine contrôle précisément chaque communication de ses entités avec les autres domaines. Il peut donc appliquer sa propre politique, et de manière centralisée (quel client contacte quel serveur de ressources, la durée de la communication et les requêtes autorisées).

L'inconvénient de cette architecture est la communication nécessaire entre les clients et les serveurs de ressources, ce qui en fait une architecture de référence. L'authentification est séparée du serveur de ressources, donc il n'a pas à retenir d'information sur tous les clients, mais il doit conserver des paramètres de session. De plus, cette communication est sécurisée avec le protocole DTLS, qui présente les inconvénients illustrés en section 3.1.11 page 22.

5.2.2 ACE OAuth

ACE OAuth est une architecture décrite dans le *draft* IETF [58] du groupe de travail ACE, comme la précédente architecture. Son objectif est de fournir un cadre logiciel pour l'autorisation dans les environnements contraints. Pour atteindre cet objectif, le *draft* IETF définit une architecture (les principales entités, leur rôle et le flux de communication) utilisant un protocole à base de tickets pour authentifier et autoriser le client. Les principales différences avec la précédente architecture sont :

1. l'usage du mécanisme OAuth 2.0 ;
2. les entités d'authentification et d'autorisation qui sont seulement présentes dans le domaine serveur.

Cette spécification permet aux environnements contraints de réutiliser le mécanisme OAuth 2.0, connu et répandu, pour un service d'autorisation.

Le fonctionnement de l'architecture peut être résumé comme suit : le client s'authentifie, puis est autorisé à accéder à la ressources et obtient un ticket, et enfin il l'utilise pour contacter le serveur de ressources. Le serveur de ressources a deux manière de valider le ticket :

1. il possède une horloge et peut exécuter rapidement des algorithmes asymétriques (vérification de signature) ;
2. il n'est pas capable de vérifier le ticket par lui-même, le ticket est alors transféré à une entité d'autorisation pour obtenir une confirmation de sa validité.

Les échanges sont basés sur le format CBOR [32], suivant les procédures décrites par COSE [31]. DTLS peut être utilisé tout en sécurisant les données grâce à COSE, en chiffrant les métadonnées, en négociant les clés de chiffrement ou en authentifiant mutuellement C et AN par exemple. ACE OAuth peut également être utilisé avec EDHOC [59], ajoutant du Perfect Forward Secrecy à COSE grâce à un échange de clés Diffie-Hellman authentifié avec des clés éphémères. ACE OAuth est utilisé avec un « profil » : mise à part l'architecture principale, des profils sont décrits pour utiliser des protocoles de communication dans l'architecture, selon les besoins liés au déploiement. Les profils décrits sont :

— OSCORE (anciennement OSCOAP) [60]
— DTLS [61]
— TLS [62]
— IPsec [63]
— EAP [64]
— MQTT [65]

Nous nous concentrerons sur les profils OSCORE et DTLS (les autres profils ne sont pas encore suffisamment matures). Dans ce chapitre nous supposons l'usage d'OSCORE avec des clés pré-configurées, un serveur d'autorisation et l'usage d'introspection (pour des messages à taille réduite).

5.2.2.1 Entités, Autorités, Domaines, Relations

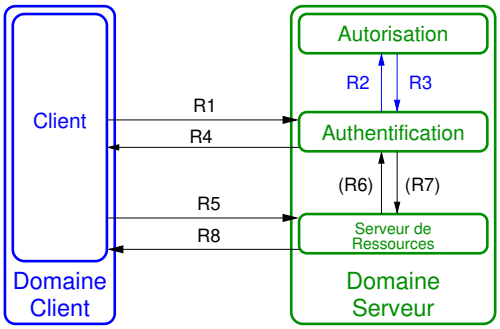


FIGURE 5.2 – Modèle de l'architecture ACE OAuth.

La figure 5.2 représente le modèle d'architecture ACE OAuth [3]. Deux domaines sont définis :
— $Domaine(client) = \{\text{Client}\}$
— $Domaine(serveur) = \{\text{Authentification, Autorisation, Serveur de Ressources}\}$
Les entités d'authentification et d'autorisation sont liées.

	entités	protocoles	propriétés	informations
R_1	C → AN	CoAP	C, I, A, R, AC	id_C, preuve id_C, requête
R_2	AN → AZ	⊥	⊥	⊥
R_3	AZ → AN	⊥	⊥	autorisation
R_4	AN → C	CoAP	C, I, A, R	id_{AN}, preuve id_{AN}, token
R_5	C → SR	CoAP	C, I, A, R	requête, token
R_6	SR → AN	CoAP	C, I, A, R	token
R_7	AN → SR	CoAP	C, I, A, R	autorisations *Autorisations = ce que le client est autorisé à faire*
R_8	SR → C	CoAP	C, I, A, R	réponse

TABLE 5.2 – Relations de l'architecture ACE OAuth.

Le tableau 5.2 et la figure 5.2 décrivent les relations de l'architecture ACE OAuth. Les relations correspondent à :

1. (R_1) le client s'authentifie auprès de l'entité d'authentification ;
2. (R_2 et R_3) les entités d'authentification et d'autorisation du domaine client se concertent pour autoriser ou non l'accès à la ressource ;
3. (R_5 et R_8) le client envoie sa requête au serveur de ressources, qui lui répond.
   Les relations R_6 et R_7 sont requises seulement si le ticket n'est pas suffisant. Dans ce cas, le ticket est une référence à la représentation d'autorisation dans le service d'authentification, que le serveur de ressources doit demander.

Les informations impliquées dans ces relations sont : I = { id, preuve d'id, token, requête, réponse }. *token* est une représentation d'autorisation, ou sa référence dans le service d'authentification, avec la signature prouvant son origine. *requête, réponse* sont des messages CoAP.

5.2.2.2 Avantages et inconvénients

Les avantages de cette architecture sont :

3. Dans le *draft* IETF ACE OAuth, le client semble faire confiance au serveur d'authentification, mais nous ne supposons pas qu'ils se font mutuellement confiance dans ce document. Nous considérons que le client peut être étranger au réseau du domaine serveur, comme dans les autres architectures.

— la séparation des entités d'authentification et d'autorisation du serveur de ressources : la cryptographie asymétrique peut être utilisée dans l'architecture, sans être exécutée sur le serveur de ressources, même si le client est étranger au réseau contraint ;
— l'architecture peut être utilisée avec des messages COSE uniquement, ce qui nécessite moins de messages qu'avec un protocole de connexion tel que DTLS.

Cette architecture a deux inconvénients. Premièrement, les en-têtes sont longues si on utilise COSE sans protocole de connexion, chaque message, en plus de transporter la demande ou la réponse, doit gérer l'authentification, gérer la négociation du matériel cryptographique, et indiquer les algorithmes utilisés. Pour nuancer, ACE OAuth peut être utilisé avec un protocole de connexion pour gérer la négociation et diminuer la taille des messages. Deuxièmement, contrairement à ACE DCAF, ACE OAuth ne peut imposer une politique centralisée dans le domaine client puisqu'il n'y a pas d'intermédiaire.

De plus, ACE OAuth est une architecture de référence, donc elle partage les mêmes avantages et les mêmes inconvénients que les autres architectures du même type, comme par exemple la protection des communications de bout-en-bout, mais que le serveur de ressources doive conserver des informations de session avec les clients.

5.2.3 Kerberos

Kerberos a été conçu dans les années 1980 au MIT pour le projet Athena, et la dernière description de cette architecture a été faite par un groupe de travail IETF en 2005 [66]. Kerberos désigne également le protocole utilisé dans la communication entre les entités.

Kerberos est une architecture intéressante puisqu'elle est la seule à dissocier explicitement les entités d'authentification, d'autorisation et le serveur de ressources. Cette architecture permet à une autorité de fournir l'authentification pour différentes applications dans le domaine serveur ou dans d'autres domaines, et jouer le rôle de tiers de confiance. L'autorisation est liée à la ressource finale, et peut être contrôlée par un autre domaine. Par exemple, une entreprise peut permettre à des employés d'une autre entreprise l'accès à ses imprimantes.

L'objectif de Kerberos est d'authentifier des utilisateurs sur un réseau non sécurisé. Kerberos est utilisé en pratique en tant que mécanisme d'authentification pour des services en ligne, comme SSH ou des serveurs de fichiers.

Pour atteindre son objectif, Kerberos définit un protocole complexe (beaucoup d'informations différentes) basé sur des tickets pour l'authentification. Un mécanisme optionnel d'autorisation explicite peut être utilisé si nécessaire pour certains serveurs de ressources. Le client demande un ticket à un service d'authentification, puis il peut demander une permission à un service d'autorisation, et finalement il contacte le serveur de ressources.

5.2.3.1 Entités, Autorités, Domaines, Relations

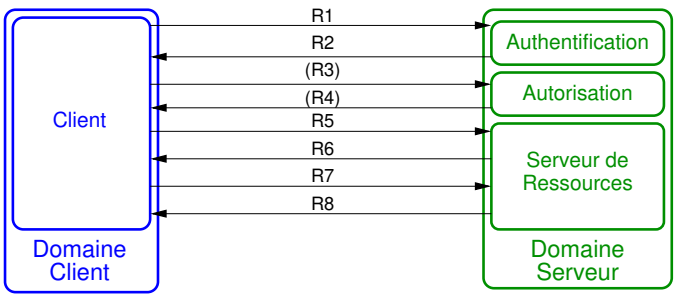


FIGURE 5.3 – Modèle de l'architecture Kerberos. Entre parenthèses, des relations optionnelles.

La figure 5.3 représente le modèle d'architecture Kerberos. Deux domaines sont définis :

— $Domaine(client) = \{\text{Client}\}$
— $Domaine(serveur) = \{\text{Authentification, Autorisation, Serveur de Ressources}\}$

Le client est authentifié par le service d'authentification qui fournit un ticket. Optionnellement, le ticket est envoyé au service d'autorisation pour demander à communiquer avec le serveur de ressources. Finalement, le client et le serveur de ressources communiquent, le client prouvant son identité (et son autorisation) avec un ticket.

	entités	protocoles	propriétés	informations
R_1	C → AN	Kerberos	-	C, AZ, n_1 *Le client envoie son identifiant et l'identifiant du service d'autorisation qu'il cherche à contacter*
R_2	AN → C	Kerberos	-	C, $TGT_{K_{AZ}}$, [AK, n_1, t_{AN}, AZ]$_{k_C}$ *Le Ticket-Granting Ticket permet au client de demander le Service Ticket du service d'autorisation*
R_3	C → AZ	Kerberos	-	$TGT_{k_{AZ}}$, [C, t_C]$_{AK}$, SR, n_2
R_4	AZ → C	Kerberos	-	C, $ST_{K_{SR}}$, [SK, n_2, t_{AZ}, SR]$_{AK}$
R_5	C → SR	Kerberos	-	$ST_{K_{SR}}$, [C, t'$_C$]$_{SK}$ *Le Service Ticket permet de demander des ressources à SR*
R_6	SR → C	Kerberos	-	[t'$_C$]$_{SK}$
R_7	C → SR	Kerberos	-	[requête]$_{SK}$
R_8	SR → C	Kerberos	-	[réponse]$_{SK}$

Table 5.3 – Relations de l'architecture Kerberos.

Le tableau 5.3 et la figure 5.3 décrivent les relations dans l'architecture Kerberos. Les relations sont :

1. (R_1 et R_2) l'entité d'authentification vérifie l'identité du client ;
2. (R_3 et R_4) l'entité d'autorisation vérifie les droits du client ;
3. (R_5 et R_6) le client établit une session avec le serveur de ressources ;
4. (R_7 et R_8) le client envoie une requête, et le serveur de ressources répond.

Kerberos peut être déployé sur un protocole de transport UDP ou TCP. Pour nous rapprocher des environnements contraints nous considérons UDP dans ce document, donc il n'y a pas de propriété de fiabilité ni d'ordonnancement dans ces relations. Kerberos définit ses propres types de messages et ses formats, donc il est considéré comme un protocole et c'est le seul impliqué dans notre exemple.

Les informations impliquées dans ces relations sont : I = {AN, AZ, C, SR, n_1, n_2, t_{AN}, t_{AZ}, t_C, t'$_C$, SK, AK, K_{SR}, K_{AZ}, ticket granting ticket (TGT), service ticket (ST), requête, réponse}. Soit :

— n_1, n_2 : nonces ;
— AN, AZ, C, SR : identités ;
— t_{AN}, t_{AZ}, t_C, t'$_C$: estampilles temporelles ;
— SK, AK : clé de session, clé d'authentification ;
— K_{SR}, K_{AZ} : clés de chiffrement ;
— ticket granting ticket : K_{AZ}+ C + t_{AN} ;
— service ticket : K_{SR}+ C + t_{AZ} ;
— requête, réponse : arbitraire.

5.2.3.2 Avantages et inconvénients

L'avantage de cette architecture est que c'est la seule à explicitement séparer les trois entités du domaine serveur (authentification, autorisation, serveur de ressources). Par conséquent, des mécanismes d'autorisation différents peuvent être intégrés selon le service offert. Kerberos permet un ensemble de règles pour des entités spécialisées avec un mécanisme unique d'authentification. Enfin, Kerberos a une gestion centralisée des identifiants.

L'inconvénient majeur de cette architecture est que les messages Kerberos sont encodés dans le format ASN.1 (Basic Encoding Rules). Ce format est non conçu pour les environnements contraints

et implique des en-têtes larges, ce qui limite l'usage de Kerberos dans ces réseaux. Par exemple, dans un réseau IEEE 802.15.4, la taille des messages est plus grande que le MTU de 127 octets [4].

De plus, Kerberos est une architecture de référence, donc elle partage les mêmes avantages et les mêmes inconvénients que les autres architectures du même type.

5.2.4 OSCAR

OSCAR (Object Security Architecture for the Internet of Things) [67] est une architecture publiée en 2015. Les objectifs de cette architecture sont de fournir une sécurité de bout-en-bout, des communications de groupe, une gestion de cache, et un trafic asynchrone pour les environnements contraints.

Pour atteindre ses objectifs, OSCAR fournit :

— un mécanisme d'autorisation explicite ;
— une clé par type de ressource (en lieu et place d'une clé par serveur de ressources), ce qui permet au client de demander une seule clé pour accéder à des ressources du même type sur différentes entités, par exemple pour demander la température dans toutes les pièces d'une maison.
— un système de chiffrement de la donnée en dehors d'une session, se rapprochant d'un objet de sécurité, ce qui permet la communication de groupe, la gestion de caches de données sécurisées et par conséquent un trafic asynchrone. Le trafic asynchrone permet aux objets de continuer de dormir tout en répondant aux requêtes des clients.

Le fonctionnement de l'architecture s'explique en deux temps. Avant la requête du client :

1. le serveur de ressources utilise un certificat pré-configuré (et des clés de cryptographie asymétrique) ;
2. puis il établit une connexion DTLS avec les entités d'authentification et d'autorisation avec ces clés ;
3. les entités d'authentification et d'autorisation fournissent des clés de chiffrement pour les ressources.

Puis, pour que le client puisse demander des ressources :

1. le client s'authentifie puis demande des clés de chiffrement aux entités d'authentification et d'autorisation ;
2. le client demande la ressource auprès du serveur de ressources ou d'une entité de cache.

5.2.4.1 Entités, Autorités, Domaines, Relations

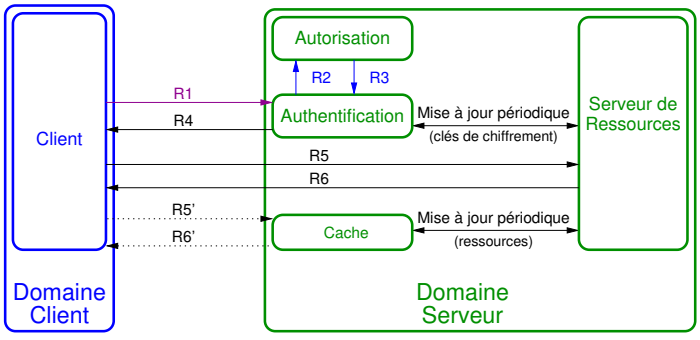


FIGURE 5.4 – Modèle de l'architecture OSCAR.

La figure 5.4 représente le modèle d'architecture OSCAR. Deux domaines sont définis :

4. Lors de la connexion le message AS_REP utilise plus de 200 octets, même avec un nom de royaume et des noms de domaines limités à 10 octets.

— $Domaine(client) = \{\text{Client}\}$
— $Domaine(serveur) = \{\text{Cache, Authentification, Autorisation, SR}\}$

Le fonctionnement global est le suivant :

1. le serveur de ressources demande une clé pour chiffrer ses ressources au service d'autorisation ;
2. le client demande cette même clé de chiffrement au service d'autorisation ;
3. le client obtient la ressource depuis le serveur de ressources ou un service de cache.

	entités	protocoles	propriétés	informations
R_1	C $\leftrightarrow$ AN	DTLS, CoAP	F, O, C, I, MA, A, R, AC	id_C, preuve id_C, id_{AN}, preuve id_{AN}, requête *preuves d'identité : non spécifié* *connexion DTLS puis requête, abrégé (lisibilité)*
R_2	AN $\rightarrow$ AZ	$\perp$	$\perp$	$\perp$
R_3	AZ $\rightarrow$ AN	$\perp$	$\perp$	autorisation
R_4	AN $\rightarrow$ C	DTLS, CoAP	C, I, MA, R	secret d'accès
R_5	C $\rightarrow$ SR	DTLS, CoAP	C, I, MA, R	$[\text{requête}]_{k_i}$ *k : clé dérivée du secret d'accès, du message et des identités des certificats* *Pas de négociation DTLS, algorithmes pré-configurés dans les certificats*
R'_5	C $\rightarrow$ Cache	DTLS, CoAP	C, I, MA, R	preuve id, requête *preuve id : clé dérivée du secret d'accès, du message et des identités des certificats* *Pas de négociation DTLS, algorithmes pré-configurés dans les certificats*
R_6	SR $\rightarrow$ C	DTLS, CoAP	C, I, MA, R	$[[\text{réponse}]_{S_{SR}}]_{K_i}$ *Signature via la clé privée du serveur de ressources* *K : clé dérivée du secret d'accès, du message et des identités des certificats*
R'_6	Cache $\rightarrow$ C	DTLS, CoAP	C, I, MA, R	$[[\text{réponse}]_{S_{SR}}]_{K_i}$ *Signature via la clé privée du serveur de ressources* *K : clé dérivée du secret d'accès, du message et des identités des certificats*

Table 5.4 – Relations de l'architecture OSCAR.

Le tableau 5.4 et la figure 5.4 décrivent les relations de l'architecture OSCAR. Les relations sont :

1. (R_1) le client s'authentifie auprès de l'entité d'authentification ;
2. (R_2 et R_3) les entités d'authentification et d'autorisation se concertent pour autoriser ou non l'accès à la ressource ;
3. (R_4) l'entité d'authentification donne la clé pour déchiffrer la ressource au client ;
4. (R_5 et R_6) le client demande la ressource au serveur de ressources, qui lui répond.
   Autre possibilité : (R'_5 et R'_6) le client demande la ressource à une entité de cache.

La cryptographie asymétrique est utilisée pour assurer l'origine des données, le secret est utilisé pour dériver une clé de chiffrement pour la réponse (ainsi que sa signature).

5.2.4.2 Avantages et inconvénients

Les avantages de cette architecture sont :
— une entité de cache est fournie ;
— le mécanisme de clés ne nécessite qu'une seule clé pour un ensemble de ressources, indépendamment du nombre de serveurs de ressources ;
— la négociation d'algorithmes ne nécessite aucun message additionnel, puisque les algorithmes sont déjà inclus dans les certificats.

L'inconvénient de cette architecture est l'usage systématique de la cryptographie asymétrique pour signer chaque ressource, ce qui implique une perte de temps à cause du calcul engendré. Par conséquent, OSCAR n'est pas compatible avec les applications à faible latence (sauf si le serveur de ressources

n'est pas contraint). De plus, la réponse contient de 40 à 44 octets de signature, ce qui limite la taille maximale de la donnée utile dans le message sur un lien avec un faible MTU.

OSCAR est une architecture de référence, mais utilise des objets de sécurité. De ce fait, un cache de données est possible en conservant une protection des données de de bout-en-bout. OSCAR se rapproche d'une architecture maître-esclave si un cache est utilisé et que les clients ne communiquent pas avec les serveurs de ressources directement.

5.2.5 MQTT-SN

MQTT-SN est une architecture basée sur MQTT, anciennement appelée *Message Queuing Telemetry Transport*, mais pour les réseaux de capteurs. La dernière version de MQTT-SN a été publiée en 2013 [19] par le groupe de travail OASIS. L'objectif est de fournir un système de publication et souscription pour la récolte de données dans les environnements contraints.

Pour atteindre son objectif, l'architecture définit un serveur mandataire, et les entités d'authentification et d'autorisation sont séparées des serveurs de ressources. Les serveurs de ressources envoient des mises à jour des données relatives à une ressource à un canal géré par le serveur mandataire, qui est suivi par les clients intéressés via un système de souscription à des canaux d'information.

MQTT-SN ne mentionne pas la sécurité dans son document de référence, cependant les communications ont de fortes chances d'être sécurisées grâce à DTLS puisque ce protocole est standard et déjà répandu dans les réseaux de capteurs. Plusieurs méthodes peuvent être utilisées pour authentifier une entité, comme des certificats, des clés publiques brutes ou des clés pré-configurées, méthodes fournies par exemple par DTLS. Nous nous basons donc sur l'utilisation de DTLS.

5.2.5.1 Entités, Autorités, Domaines, Relations

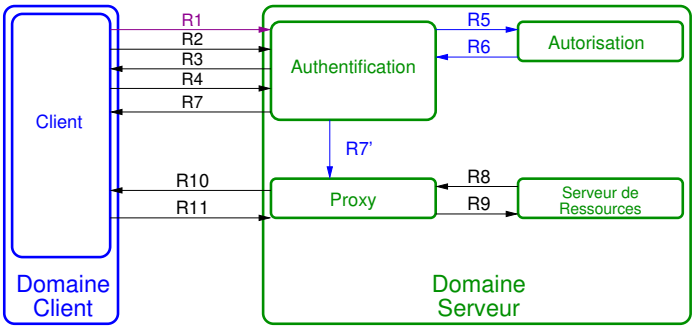


FIGURE 5.5 – Modèle de l'architecture MQTT-SN.

La figure 5.5 représente le modèle d'architecture MQTT-SN. Deux domaines sont définis :

— $Domaine(client) = \{\text{Client}\}$

— $Domaine(serveur) = \{\text{Proxy, Authentification, Autorisation, Serveur de Ressources}\}$

La séparation entre le serveur de ressources et les autres entités implique qu'un autre appareil peut gérer les calculs lourds (le serveur mandataire). Cependant, le mécanisme de publication et souscription implique des mises à jour régulières, le serveur de ressources reste actif même lorsqu'il n'y a pas de clients.

Le tableau 5.5 et la figure 5.5 décrivent les relations de l'architecture MQTT-SN. Les informations impliquées dans ces relations (mises à part celles de DTLS) sont : I = {keep alive, id protocole, id message, drapeaux, id sujet ou nom sujet, donnée}

— id message : identifiant du message ;

— (id, nom) sujet : identifiant de l'élément d'intérêt auquel les clients souscrivent, par exemple le nom d'une ressource d'un capteur particulier tel que `"/sensorA/temp"` ;

	entités	protocoles	propriétés	informations
R_1	C ↔ AN	DTLS	F, O, C, I, MA, A, R, AC	id_C, preuve id_C, id_{AN}, preuve id_{AN} *connexion DTLS abrégée (lisibilité)*
R_2	C → AN	DTLS, MQTT-SN	C, I, MA, R	id_C, keep alive, id protocole, drapeaux
R_3	AN → C	DTLS, MQTT-SN	C, I, MA, R	code de retour
R_4	C → AN	DTLS, MQTT-SN	C, I, MA, R	id message, (id, nom) sujet *Le client souscrit à un sujet*
R_5	AN → AZ	⊥	⊥	⊥
R_6	AZ → AN	⊥	⊥	autorisation *L'entité d'autorisation permet la souscription*
R_7	AN → C	DTLS, MQTT-SN	C, I, MA, R	id message, id sujet, code de retour
R'_7	AN → Proxy	DTLS, MQTT-SN	⊥	⊥ *Le client a souscrit au sujet*
R_8	SR → Proxy	DTLS, MQTT-SN	C, I, MA, R	id sujet, id message, donnée
R_9	Proxy → SR	DTLS, MQTT-SN	C, I, MA, R	id sujet, id message, code de retour
R_{10}	Proxy → C	DTLS, MQTT-SN	C, I, MA, R	id sujet, id message, donnée
R_{11}	C → Proxy	DTLS, MQTT-SN	C, I, MA, R	id sujet, id message, code de retour

TABLE 5.5 – Relations de l'architecture MQTT-SN.

— keep alive : durée de maintien de la connexion ;
— donnée : donnée MQTT-SN, mise à jour d'un capteur.

5.2.5.2 Avantages et inconvénients

Les avantages de cette architecture sont :
— la conception de MQTT-SN adaptée aux les réseaux contraints : les messages ont une petite taille, le serveur de ressources ne se connecte qu'à un seul appareil (le routeur qui fait office de proxy) [5] ;
— le routeur gère les clients et leurs requêtes : les clients et les serveurs de ressources ne se parlent donc pas directement, et les serveurs de ressources n'ont pas à connaître les clients (ce qui représente un gain de mémoire) ;
— le routeur se contente de recevoir les mises à jour des données, il n'envoie pas de requête au serveur de ressources. Il n'y a donc pas de corrélation entre le nombre de clients et le nombre de messages envoyés par le serveur de ressources.

MQTT-SN présente plusieurs inconvénients. Premièrement, l'usage de DTLS implique une dizaine de messages à la connexion, puis 29 octets dans chaque message, voire la section 3.1.11 page 22. Cependant, cette connexion peut être persistante entre le serveur de ressources et le routeur. Deuxièmement, l'absence de métadonnées dans le protocole MQTT implique que les clients ne peuvent pas demander un encodage spécial d'une ressource. Par conséquent les capteurs doivent avoir des représentations identiques, comme par exemple une représentation en chaîne de caractères d'une température en degrés Celsius. Enfin, cette architecture est adaptée à la remontée d'informations de capteurs, mais ne permet pas de manipuler des actionneurs.

MQTT-SN est une architecture maître-esclave, les clients ne communiquent pas directement avec les serveurs de ressources.

5. Dans la documentation, le routeur fait l'interface entre le serveur de ressources et un courtier (*broker*) qui gère les clients. Nous considérons dans ce document que le routeur MQTT-SN est également le courtier, dans le but de simplifier la description de l'architecture.

5.2.6 A-OSCORE : Architecture basée sur OSCORE

OSCORE (anciennement OSCOAP, Object Security of CoAP) [68] est un mécanisme de sécurité basé sur COSE et CoAP dont l'objectif est de protéger les données de bout-en-bout. A-OSCORE représente l'architecture basée sur le protocole OSCORE.

Cette architecture simple permet une authentification (basée sur la connaissance d'une clé) et une autorisation (avec des règles statiques basées sur l'identité du pair) basiques. Dans notre architecture :

— OSCORE n'utilise que CoAP et COSE[6] (format et procédures) ;
— l'autorisation est implicite (un client authentifié a accès à tout) ;
— cette architecture protège la charge utile d'un message CoAP et ses options qui ne sont pas liées au transfert du message.

5.2.6.1 Entités, Autorités, Domaines, Relations

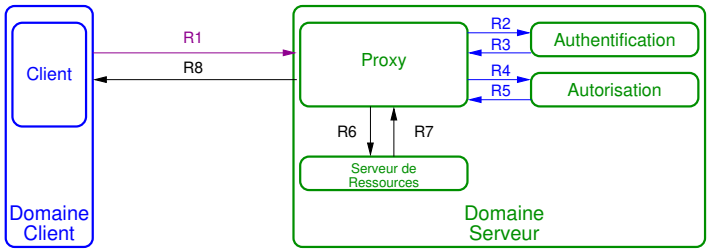


FIGURE 5.6 – Modèle d'une architecture basée sur OSCORE.

La figure 5.6 représente le modèle de l'architecture OSCORE. Deux domaines sont définis :

— $Domaine(client) = \{\text{Client}\}$
— $Domaine(serveur) = \{\text{Proxy, Authentification, Autorisation, Serveur de Ressources}\}$

	entités	protocoles	propriétés	informations
R_1	C → Proxy	CoAP	C, I, MA, R	$id_{C_{proxy}}$, $Seq_{C_{proxy}}$, requête requête = URI AS, URI ressource, action
R_2	Proxy → AN	⊥	⊥	⊥
R_3	AN → Proxy	⊥	⊥	⊥
R_4	Proxy → AZ	⊥	⊥	⊥
R_5	AZ → Proxy	⊥	⊥	autorisation
R_6	Proxy → SR	CoAP	C, I, MA, R	$id_{proxy_{SR}}$, $Seq_{proxy_{SR}}$, requête
R_7	SR → Proxy	CoAP	C, I, MA, R	$id_{SR_{proxy}}$, $Seq_{SR_{proxy}}$, réponse
R_8	Proxy → C	CoAP	C, I, MA, R	id_{proxy_C}, Seq_{proxy_C}, réponse

TABLE 5.6 – Relations de l'architecture A-OSCORE.

Le tableau 5.6 et la figure 5.6 décrivent les relations dans l'architecture OSCORE. Le premier message est déjà chiffré puisque l'échange de clés est déjà effectué. Les informations impliquées dans ces relations sont : $I = \{id_{a_b}, Seq_{a_b}, \text{requête, réponse}\}$

— id_{a_b} est un identifiant de a tel que connu par b ;
— Seq_{a_b} est un numéro de séquence (de a à b) ;
— *requête* et *réponse* sont des messages CoAP.

5.2.6.2 Avantages et inconvénients

L'avantage de cette architecture est le peu d'échanges de messages, car il n'y a pas de poignée de main obligatoire et l'échange de clés est antérieure à la requête.

6. COSE permet la sécurité de bout-en-bout de CoAP avec des services de sécurité basiques : confidentialité, authentification et intégrité des messages, et une protection contre le rejeu.

L'architecture présente deux inconvénients. Premièrement, le manque de protection contre le déni de service : des paquets chiffrés forgés peuvent être envoyés pour surcharger le serveur de ressources. Deuxièmement, l'architecture ne fournit pas de négociation des algorithmes de chiffrement. La négociation des algorithmes ne fait pas partie du document OSCORE, mais un *draft* IETF fournit cette négociation en utilisant un échange de clés Diffie-Hellman avec des clés éphémères [59].

A-OSCORE est une architecture de référence car la protection des communications se passe de saut en saut, mais la donnée est chiffrée de bout-en-bout.

5.2.7 A-DTLS : Architecture basée sur DTLS

A-DTLS représente l'architecture basée sur le protocole DTLS. Nous avons créé des architectures minimales sur des protocoles de sécurité en les utilisant entre deux appareils (le client et le serveur de ressources), ce qui implique que l'appareil du serveur de ressources contient également les entités d'authentification et d'autorisation. L'architecture A-DTLS n'est pas décrite dans la littérature : nous la définissons puisqu'elle a des chances d'être déployée en pratique. Notre objectif est de formaliser sa description et l'inclure dans notre comparaison.

Dans cette architecture, seulement deux appareils (le client et le serveur de ressources) échangent des informations en utilisant le protocole DTLS. Avec cette architecture, les entités d'authentification et d'autorisation sont embarquées dans le serveur de ressources. De plus, ces entités fournissent des services basiques tels que :

1. l'authentification basée sur la connaissance d'un secret partagé ;
2. l'autorisation implicite utilisant des règles statiques ;
3. des rôles basés sur l'identité du pair.

5.2.7.1 Entités, Autorités, Domaines, Relations

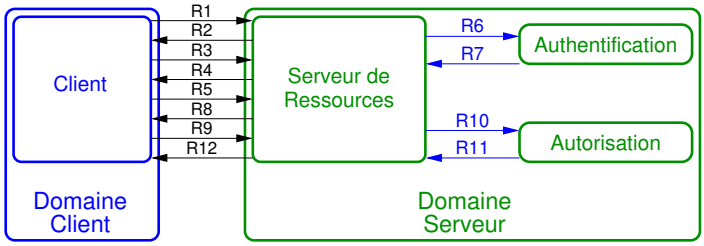


FIGURE 5.7 – Modèle d'une architecture basée sur DTLS.

La figure 5.7 représente cette architecture. Deux domaines sont définis :
— $Domaine(client) = \{\text{Client}\}$
— $Domaine(serveur) = \{\text{Authentification, Autorisation, Serveur de Ressources}\}$
Le tableau 5.7 et la figure 5.7 décrivent les relations :

1. (R_1 à R_8) l'authentification et la négociation des algorithmes ;
2. (R_9 à R_{12}) requête du client, autorisation de l'entité d'autorisation, puis réponse du serveur de ressources.

5.2.7.2 Avantages et inconvénients

A-DTLS est une architecture de référence, basée sur le protocole DTLS, sans ajout d'un nœud intermédiaire. De ce fait, les avantages et inconvénients de cette architecture sont liés à ce protocole, comme illustré en section 3.1.11 page 22.

	entités	protocoles	propriétés	informations
R_1	C → SR	DTLS		id_C, version, id session, aléa 1, algorithmes
R_2	SR → C	DTLS		cookie *SR envoie un cookie, reste sans état (protection DoS)*
R_3	C → SR	DTLS	I, MA, R, AC	id_C, version, id session, aléa 1, algorithmes, cookie
R_4	SR → C	DTLS	I, MA, R, AC	version, algorithmes retenus, aléa 2, (certificat, paramètres DH, types de certificats acceptés et autorités,) id_{SR}, preuve id_{SR}
R_5	C → SR	DTLS	I, MA, R, AC	id_C, preuve id_C, (certificat, signature, paramètres DH,) [données] chiffrées comme négocié *Ce message vérifie que la négociation s'est bien passée* *(en vérifiant l'intégrité et l'authenticité des anciens messages)*
R_6	SR → AN	⊥	⊥	⊥
R_7	AN → SR	⊥	⊥	⊥
R_8	SR → C	DTLS	C, I, MA, R, AC	[données] chiffrées comme négocié *Ce message vérifie que la négociation s'est bien passée* *(en vérifiant l'intégrité et l'authenticité des anciens messages)*
R_9	C → SR	DTLS, CoAP	C, I, MA, R	requête
R_{10}	SR → AZ	⊥	⊥	⊥
R_{11}	AZ → SR	⊥	⊥	autorisation
R_{12}	SR → C	DTLS, CoAP	C, I, MA, R	réponse

TABLE 5.7 – Relations de l'architecture basée sur DTLS. Informations optionnelles entre parenthèses.

5.2.8 A-IPsec : Architecture basée sur IPsec

A-IPsec représente l'architecture basée sur les protocoles IPsec. IPsec [69] est un ensemble de protocoles, parfois décrit comme une solution alternative viable pour l'Internet des Objets [70]. Notre architecture est définie par deux appareils (le client et le serveur de ressources) échangeant des informations utilisant le protocole IP Encapsulating Security Payload (ESP) [71] [72] pour protéger la charge utile de la couche réseau [7].

Ces protocoles apportent à notre architecture la confidentialité gérée par flots de trafic, il est possible de gérer différemment deux flots de données (avec différents algorithmes, ou différentes clés par exemple). Ces protocoles permettent d'authentifier des pairs, de protéger la communication et de vérifier l'origine de la donnée.

Notre architecture utilise également Internet Key Exchange (v2) [74] [75] pour la gestion des clés et l'agilité cryptographique [8]. Comme avec l'architecture basée sur DTLS, les entités d'authentification et d'autorisation sont embarquées dans le serveur de ressources, fournissant des services basiques.

5.2.8.1 Entités, Autorités, Domaines, Relations

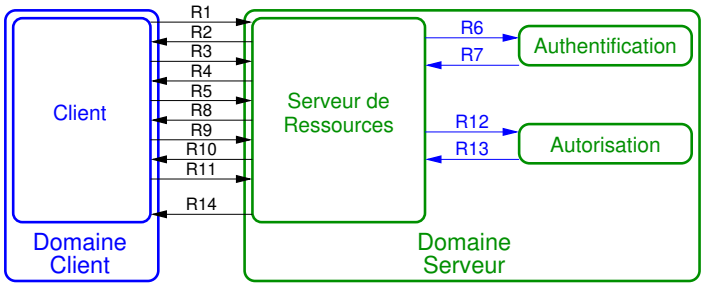


FIGURE 5.8 – Modèle d'une architecture basée sur IPsec.

Le schéma 5.8 présente notre architecture de sécurité basée sur IPsec. Deux domaines sont définis :

7. Le protocole Authentication Header (AH) [73] [72] peut également authentifier l'en-tête IP, mais quelques inconvénients (comme la traversée de NAT) et le manque de cas d'usage comparé à ESP limitent son utilisation en pratique.

8. Le protocole IPsec n'est pas lié à IKE, d'autres protocoles peuvent être utilisés à la place.

— $Domaine(client) = \{\text{Client}\}$
— $Domaine(serveur) = \{\text{Authentification, Autorisation, Serveur de Ressources}\}$

	entités	protocoles	propriétés	informations
R_1	C → SR	IKEv2		SA_C, KE_C, N_C *SA : association de sécurité* *KE : échange de clé* *N_C : nonce du client* *Premier message, le client se connecte à SR*
R_2	SR → C	IKEv2		cookie *SR envoie un cookie, reste sans état (protection DoS)*
R_3	C → SR	IKEv2		cookie, SA_C, KE_C, N_C *C envoie un cookie en réponse au message original*
R_4	SR → C	IKEv2		SA, KE, N_{SR} *N_{SR} : nonce du serveur de ressources* *SR envoie les algorithmes à utiliser*
R_5	C → SR	IKEv2		id_C, AUTH, SA, TS_C, TS_{SR} *TS : Sélecteur de trafic (choix de ce qui sera chiffré)* *C s'authentifie auprès de SR*
R_6	SR → AN	⊥	⊥	⊥
R_7	AN → SR	⊥	⊥	⊥
R_8	SR → C	IKEv2		id_{SR}, AUTH, SA, TS_C, TS_{SR} *SR s'authentifie auprès de C*
R_9	C → SR	IKEv2	C, I, MA, R	SA, SA, N_C, (KE_C,) TS_C, TS_{SR} *Premier message chiffré, pour créer une association de sécurité*
R_{10}	SR → C	IKEv2	C, I, MA, R	SA, N_{SR}, [KE_{SR},] TS_C, TS_{SR} *Réponse de SR, création d'une nouvelle association de sécurité*
R_{11}	C → SR	IKEv2, CoAP	C, I, MA, R	requête
R_{12}	SR → AZ	⊥	⊥	⊥
R_{13}	AZ → SR	⊥	⊥	autorisation
R_{14}	SR → C	IKEv2, CoAP	C, I, MA, R	réponse

TABLE 5.8 – Relations de l'architecture basée sur IPsec (IKEv2). Informations optionnelles entre parenthèses.

Le tableau 5.8 et la figure 5.8 décrivent les relations de l'architecture A-IPsec. Les relations sont :

1. (R_1 et R_2) le client démarre une connexion, le serveur de ressources lui envoie un cookie ;
2. (R_3 et R_4) le client renvoie le cookie et la négociation des algorithmes est faite ;
3. (R_5 à R_8) le client et le serveur de ressources s'authentifient l'un à l'autre ;
4. (R_9 et R_{10}) messages chiffrés, pour créer une session chiffrée pour le trafic sélectionné ;
5. (R_{11} à R_{14}) requête du client, autorisation de l'entité d'autorisation, puis réponse du serveur de ressources.

5.2.8.2 Avantages et inconvénients

A-IPsec a plusieurs avantages. Premièrement, l'architecture permet un réglage fin de la protection des communications entre deux points. Chaque communication peut être protégée avec un secret et des algorithmes différents, avec une discrimination des communications sur l'adresse IP, les ports ou bien le protocole de couche supérieure. Deuxièmement, le protocole IPsec est partie intégrante du standard IPv6. Par conséquent, tous les périphériques compatibles IPv6 doivent l'implémenter en suivant les recommandations de l'IETF [76], les architectures basées sur IPsec n'ont alors besoin que d'une implémentation standard d'IPv6 [9].

Cette architecture présente toutefois deux inconvénients. Premièrement, le protocole IPsec est complexe, sa complexité est due aux possibilités offertes par les réglages fins qui ont été étudiés pour des communications sur Internet. Cette finesse de réglage est une fonctionnalité non pertinente dans des réseaux contraints où il est important de limiter jusqu'à la taille du binaire exécutable du

9. Cependant, l'organisation IETF reconnaît que des alternatives à IPsec peuvent être justifiées dans les environnements contraints (comme DTLS).

programme, qui doit être fortement épuré. Deuxièmement, cette architecture ne permet pas à plusieurs clients d'être sur le même périphérique ; le protocole IPsec ne permet pas de discriminer les canaux de communication sur l'identité du client, mais seulement sur son adresse IP.

A-IPsec est une architecture de référence, les clients communiquent directement avec les serveurs de ressources.

5.2.9 A-SSH : Architecture basée sur SSH

SSH (Secure SHell) [77] est un ensemble de protocoles [10], ces protocoles sont simples et largement déployés. Notre architecture A-SSH est basée sur ces protocoles.

Tel qu'énoncé au début du chapitre, nous avons créé des architectures minimales sur des protocoles de sécurité en les utilisant entre deux appareils (le client et le serveur de ressources), ce qui implique que l'appareil du serveur de ressources contient également les entités d'authentification et d'autorisation. Cette architecture simple permet une authentification (basée sur la connaissance d'une clé) et une autorisation (avec des règles statiques basées sur l'identité du pair) basiques. En plus des méthodes d'authentification par clé publique ou par mot de passe, cette architecture permet également l'authentification basée sur l'hôte.

5.2.9.1 Entités, Autorités, Domaines, Relations

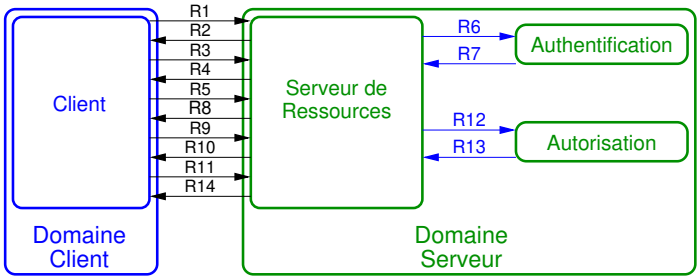


FIGURE 5.9 – Modèle d'une architecture basée sur SSH.

La figure 5.9 représente une architecture de sécurité minimale basée sur SSH. Deux domaines sont définis :

— $Domaine(client) = \{\text{Client}\}$
— $Domaine(serveur) = \{\text{Authentification, Autorisation, Serveur de Ressources}\}$

Le tableau 5.9 et la figure 5.9 décrivent les relations dans l'architecture minimale basée sur SSH. Les relations sont :

1. (R_1 à R_8) négociation des algorithmes, authentification ;
2. (R_9 et R_{10}) création d'un *canal* pour laisser passer les données, ce mécanisme permet différents flots de données dans une seule connexion ;
3. (R_{11} à R_{14}) requête, autorisation, réponse.

SSH utilise TCP pour gérer la fiabilité et l'ordre d'arrivée des paquets. Les entités d'authentification, d'autorisation et le serveur de ressources sont liés.

10. Premièrement, SSH spécifie une couche transport [78] pour fournir un chiffrement de bout-en-bout des communications, une négociation d'algorithmes, l'intégrité, la confidentialité, l'échange de clés et l'authentification des messages. Deuxièmement, SSH spécifie un protocole d'authentification des pairs [79]. Finalement, SSH spécifie un protocole de connexion [80] pour fournir des canaux multiplexés de transport, et des fonctionnalités avec des objectifs spécifiques comme l'exécution de commandes à distance.

	entités	protocoles	propriétés	informations
R_1	C → SR	TCP, SSH	F, O	version
R_2	SR → C	TCP, SSH	F, O	version
R_3	C → SR	TCP, SSH	F, O	id_C, $algorithmes_C$, $langue_C$
R_4	SR → C	TCP, SSH	F, O	id_{SR}, $algorithmes_{SR}$, $langue_{SR}$
R_5	C → SR	TCP, SSH	F, O AC	paramètres DH, signature
R_6	SR → AN	⊥	⊥	⊥
R_7	AN → SR	⊥	⊥	⊥
R_8	SR → C	TCP, SSH	F, O, AC	paramètres DH, signature
R_9	C → SR	TCP, SSH	F, O, C, I, MA, R	ouverture de canal
R_{10}	SR → C	TCP, SSH	F, O, C, I, MA, R	confirmation de l'ouverture du canal
R_{11}	C → SR	TCP, SSH, CoAP	F, O, C, I, MA, R	requête
R_{12}	SR → AZ	⊥	⊥	⊥
R_{13}	AZ → SR	⊥	⊥	autorisation
R_{14}	SR → C	TCP, SSH, CoAP	F, O, C, I, MA, R	réponse

TABLE 5.9 – Relations de l'architecture basée sur SSH.

5.2.9.2 Avantages et inconvénients

A-SSH n'a pas d'avantage par rapport aux architectures présentées précédemment et dans un contexte d'environnements contraints. Cette architecture a été étudiée parce que le protocole SSH est très répandu et qu'il est probable qu'une architecture basée sur ce protocole soit déployée en pratique. Le protocole SSH est pensé pour quelques usages précis sur des réseaux non contraints.

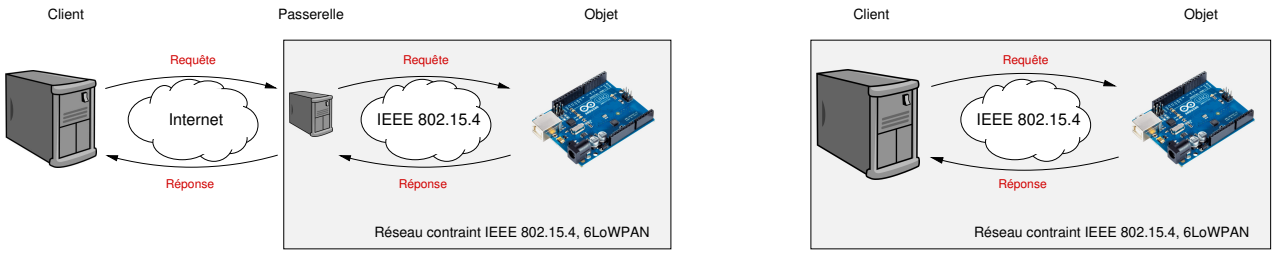
L'architecture A-SSH présente deux inconvénients. D'une part, la négociation s'effectue avec les algorithmes décrits textuellement, ce qui n'est pas adapté aux réseaux contraints. Par exemple, les algorithmes sont décrits par des chaînes de caractères tels que « `aes128-gcm@openssh.com` » ce qui donne des messages trop longs pour des réseaux contraints. D'autre part, l'usage de TCP permet de fragmenter le message, mais cela amène un autre inconvénient majeur puisque TCP consomme 20 octets de charge utile par message.

A-SSH est une architecture de référence, les clients communiquent directement avec les serveurs de ressources.

5.3 Comparaison

Dans cette section, nous détaillons un contexte (deux scénarii et une pile de protocoles) ainsi que des critères sur lesquels comparer les architectures, puis nous présentons les résultats.

5.3.1 Scénarii



(A) Client externe au réseau contraint. (B) Client interne au réseau contraint.

FIGURE 5.10 – Scénarii.

La figure 5.10 présente les scénarii utilisés pour la comparaison des architectures. Ils couvrent les mécanismes d'authentification utilisés couramment en pratique (clés pré-configurées, certificats), avec un client étranger ou local au domaine serveur.

Dans le premier scénario, une entité cliente étrangère et sans contrainte fait une demande au serveur de ressources (contraint) dans le domaine serveur. L'architecture a besoin d'une authentification mutuelle entre le client et l'entité d'authentification, d'une entité d'autorisation pour fournir des permissions (et les opérations autorisées) pour la requête, et de la protection des communications (confidentialité, authenticité et intégrité des messages). Dans la plupart des cas, l'authentification (et l'autorisation) est réalisée grâce à la cryptographie asymétrique (une paire de clés publique et privée de chaque côté), avec un algorithme d'authentification par signature numérique avec courbes elliptiques (ECDSA). La communication est sécurisée par cryptographie symétrique (nous détaillerons cela en section 5.3.2). La clé pour protéger la communication est fournie par ephemeral elliptic curve Diffie-Hellman (ECDHE) quand cela est possible. Les certificats et l'algorithme Diffie-Hellman permettent au serveur de ressources de ne pas avoir à retenir d'informations à propos de chaque client donc cela est utile quand il y a un grand nombre de clients.

Dans le second scénario, l'entité cliente est sur un appareil contraint dans le domaine serveur, et l'authentification par clé pré-configurée est préférée. Chaque serveur de ressources est contacté par un nombre limité de clients, ce qui permet aux serveurs de ressources de garder une clé partagée pour chacun d'eux, et vice versa.

Certaines architectures n'ont pas de mécanismes pour le premier scénario, comme par exemple Kerberos puisqu'il n'a pas été initialement conçu pour gérer de cryptographie asymétrique. D'autres ne sont pas adaptées pour le MTU de IEEE 802.15.4, comme par exemple A-OSCORE dans le premier scénario, avec son format de messages basé sur COSE.

5.3.2 Pile de communication

Notre définition d'une architecture de sécurité n'est pas liée à une technologie particulière de niveau 2, à un réseau ou encore à une couche transport. Pour la comparaison, ces paramètres sont fixés et basés sur les standards d'environnements contraints, sauf si spécifié autrement dans l'architecture.

Les couches physique et liaison utilisées dans la comparaison sont celles du standard IEEE 802.15.4. Cette technologie ne permet qu'une taille de messages de 127 octets, dont un en-tête de couche liaison de 9 octets, laissant peu de place pour la charge utile. La couche réseau utilisée pour cette comparaison est 6LoWPAN, ce qui laisse 116 octets pour le reste. La couche transport utilisée est UDP quand cela est possible (sinon TCP), soit 112 octets pour le reste du message (96 avec TCP). La couche applicative utilisée pour la comparaison est CoAP. L'en-tête de CoAP occupe 4 octets, et deux de plus sont utilisés pour préciser le format de la donnée via l'option `Content-Format`.

Nous avons choisi le chiffrement par AES 128 bits avec le mode de chiffrement CCM avec 8 octets de valeur d'intégrité (ICV) pour la sécurité des communication dans cette comparaison. Kerberos et l'architecture basée sur SSH ne possèdent pas cet algorithme. SSH peut être utilisé avec un autre algorithme AEAD tel qu'AES 128 GCM [81], mais il n'y a pas d'algorithme AEAD pour Kerberos. Kerberos est donc utilisé avec AES 128 bits avec le mode Ciphertext Stealing (CTS) et SHA256 pour la vérification d'intégrité, ce qui requiert 16 octets.

Le schéma 5.11 montre la pile de communication utilisée dans notre comparaison sans sécurité. Dans quelques architectures, certaines de ces couches peuvent changer : par exemple, MQTT-SN utilise sa propre couche applicative.

5.3.3 Résultats

La comparaison se fait sur la sécurité, les contraintes et les aspects généraux des architectures, tels que décrits dans la suite et discutés dans la section suivante. Les tableaux 5.10 et 5.11 montrent un résumé.

	OSCAR	ACE-DCAF	ACE-OAuth	Kerberos
entités du domaine client	client	client, AN, AZ	client	client
entités du domaine serveur	SR, AN, AZ, Proxy	SR, AN, AZ	SR, AN, AZ	SR, AN, AZ
Sécurité des communications				
protection contre rejeu	oui	oui	oui (DTLS et/ou COSE)	oui
protection contre le DoS	**non**	cookie (DTLS)	**non** (sauf si DTLS)	**non**
chiffrement métadonnées	**non**	oui	**non** (sauf si DTLS)	**non**
agilité cryptographique	**non** algorithmes indiqués dans les certificats	oui	**non** (sauf si DTLS)	oui
Éléments de sécurité				
crypto. asymétrique (SR)	**oui**	non	non (sauf si DTLS)	non
autorisation explicite	oui	oui	oui	oui
granularité d'autorisation	type de ressource	ressource	ressource	**SR**
gestion automatique des clés	oui	oui	oui	oui
contexte par client (SR)	non	**oui**	**oui** (DTLS et/ou COSE)	**oui**
Contraintes				
nb relations entre domaines	4 + DTLS	4 + DTLS + (D)TLS (+TCP)	4 + DTLS	6
horloge requise (SR)	**oui**	non	non	**oui**
service de cache de données	oui	**non**	**non**	**non**
surcharge taille de messages	**grande**, signature (40-44 octets)	**moyen**, DTLS (29 octets)	**grande**, COSE (+ DTLS) (> 20 + 29 octets)	**très grande** **> 127 octets à la connexion**
nb msg 1ère connexion (SR)	0 (Proxy)	10 (nb min msg pour DTLS)	2-4 (COSE, non EDHOC) 12-16 (DTLS + introspect)	4
nb msg 1ère connexion (C)	2	12 (DTLS + autorisation)	4 (sans DTLS)	4
nb msg chaque requête (SR)	2	2	2-4 (sans DTLS)	2
nb msg chaque requête (C)	2	2	2-4 (sans DTLS)	2
connexions maintenues (SR)	1-2 (AN et Proxy)	1 (AN+AZ)	0-1 (introspection)	0
nombre de messages (SR)	**dépend de la fréquence des màj**	**dépend du nb de clients et fréquence des requêtes**	**dépend du nb de clients et fréquence des requêtes**	**dépend du nb de clients et fréquence des requêtes**
Aspects généraux				
protocoles utilisés	DTLS + CoAP	(D)TLS (+ TCP) + CoAP	OSCORE (+ DTLS)	Kerberos (+ (D)TLS)
arch avec app spécifique	**oui**	**oui**	**oui**	non
découverte de ressources	peut être implémenté sur AN	**non**	**non**	**non**

TABLE 5.10 – Comparaison (1/2) des architectures sur des aspects de sécurité et de contraintes. En vert les avantages, en rouge les inconvénients (ou un résultat dépendant du déploiement).

	MQTT-SN	A-DTLS	A-IPsec	A-SSH	A-OSCORE
entités du domaine client	client	client	client	client	client
entités du domaine serveur	SR, AN, AZ, Proxy	SR, AN, AZ	SR, AN, AZ	SR, AN, AZ	SR, AN, AZ, Proxy
Sécurité des communications					
protection contre rejeu	oui	oui	oui	oui	oui
protection contre le DoS	cookie	cookie	cookie	oui (TCP)	**none**
chiffrement métadonnées	oui	oui	oui	oui	**non**
agilité cryptographique	oui	oui	oui	oui	**non**
Éléments de sécurité					
crypto. asymétrique (SR)	non	oui *	oui *	**oui**	oui *
autorisation explicite	oui	**non**	**non**	**non**	**non**
granularité d'autorisation	ressource	**SR**	**SR**	**SR**	**SR**
gestion automatique des clés	oui	oui	oui	oui	oui
contexte par client (SR)	non	**oui**	**oui**	**oui**	**oui**
Contraintes					
nb relations entre domaines	6 + DTLS	8	10	10 + TCP	2
horloge requise (SR)	non	oui *	oui *	oui *	**oui ***
service de cache de données	oui	**non**	**non**	**non**	**non**
surcharge taille de messages	**grande** DTLS (29 octets)	**grande** DTLS (29 octets)	**moyenne** ESP (18 octets sans padding)	**grande** TCP, SSH (20 + 20 octets)	**très grande** **> 127 octets au scénario 1**
nb msg 1ère connexion (SR)	14	14 ** (10)	8	11 (TCP + SSH)	2
nb msg 1ère connexion (C)	18 ** (14)	14 ** (10)	8	11 (TCP + SSH)	2
nb msg à chaque requête (SR)	0 (asynchrone) 1 à 4 / màj	2	2	2	2
nb msg à chaque requête (C)	0 (asynchrone) 1 à 2 / màj	2	2	2	2
connexions maintenues (SR)	1 (AN+AZ+Proxy)	0	0	0	0
surcharge de messages (SR)	**dépend de la fréquence des màj**	**dépend du nb de clients et fréquence des requêtes**	**dépend du nb de clients et fréquence des requêtes**	**dépend du nb de clients et fréquence des requêtes**	**dépend du nb de clients et fréquence des requêtes**
Aspects généraux					
protocoles utilisés	DTLS + MQTT-SN	DTLS + CoAP	IPsec + CoAP	TCP + SSH + CoAP	CoAP (+ DTLS)
arch avec app spécifique	**oui**	non	non	non	non
découverte de ressources	**non**	**non**	**non**	**non**	**non**

TABLE 5.11 – Comparaison (2/2) des architectures sur des aspects de sécurité et de contraintes. En vert les avantages, en rouge les inconvénients (ou un résultat dépendant du déploiement). * change au scénario 2, ** plus les paquets pour échanger les certificats.

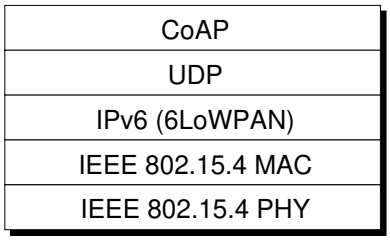


FIGURE 5.11 – Pile de protocoles généralement utilisée pour la comparaison.

5.3.3.1 Sécurité

Le premier critère est la protection des communications.

Au delà des algorithmes cryptographiques, des mécanismes complémentaires interviennent pour protéger les communications, comme :

— la vérification du rejeu ;
— la protection contre le déni de service ;
— le chiffrement des métadonnées ;
— l'agilité cryptographique.

IPsec chiffre tout au-dessus de la couche réseau, SSH et toute architecture basée sur DTLS (ACE DCAF, MQTT-SN...) chiffrent au-dessus de la couche transport, et les autres architectures comme OSCAR ne chiffrent que la charge utile de la couche applicative.

Le critère *cryptographie asymétrique sur SR* montre les architectures dans lesquelles le serveur de ressources doit faire de longs calculs, comme de l'authentification par certificats. Par exemple dans l'architecture OSCAR chaque représentation de la ressource est signée par le serveur de ressources, et dans notre architecture basée sur DTLS l'authentification est effectuée par le serveur de ressources ce qui implique l'usage de cryptographie asymétrique dans le premier scénario à cause de l'usage de certificats et de l'algorithme Diffie-Hellman. Quand une architecture oblige l'usage de cryptographie asymétrique, la couleur rouge est utilisée, quand ce n'est pas le cas la couleur verte est utilisée, enfin si le choix est lié aux besoins du déploiement le texte est en noir.

L'*autorisation* est explicite si l'architecture définit spécifiquement les messages (ou les informations) pour le mécanisme d'autorisation (avec un serveur d'autorisation dédié par exemple). Sinon, l'architecture fournit seulement une autorisation implicite ou délègue ce mécanisme à la couche applicative. Selon la *granularité de l'autorisation*, l'accès peut être accordé à un serveur de ressources, à un type de ressources ou directement à une ressource précise sur un serveur de ressources.

La *gestion automatique des clés* peut être atteinte avec l'algorithme Diffie-Hellman ou une clé partagée. Par exemple, le protocole DTLS utilise des nombres aléatoires en combinaison avec des clés pré-configurées pour créer de nouvelles clés de session à chaque connexion, sans nécessiter de longs calculs, dans le second scénario. Toutes les architectures ont une gestion automatique des clés.

Garder un *contexte par client* (sur le serveur de ressources) permet une association de sécurité unique entre deux entités. Un tel contexte est composé de numéros de séquence, des algorithmes négociés, des clés effectives de chiffrement et d'autres informations requises pour la communication sécurisée. Il n'y a pas de contexte entre le client et le serveur de ressources dans OSCAR puisque la clé de chiffrement d'un type de ressources est le même pour tous les clients. L'architecture MQTT-SN a un serveur d'authentification dédié sur un périphérique non contraint, allégeant le coût de l'authentification sur le serveur de ressources qui n'a désormais besoin que de s'authentifier auprès du serveur d'authentification.

La *sécurité de bout-en-bout* est intéressante pour limiter la surface d'attaque au serveur de ressources dans le domaine serveur, mais nécessite plus de mémoire (un contexte par client).

5.3.3.2 Contraintes

Le *nombre de relations entre les domaines* aide à établir un ordre de grandeur du nombre de messages échangés entre le client et le domaine serveur. Cela est lié principalement aux échanges entre deux appareils, montrant le nombre minimal de messages échangés pour une requête (les messages échangés dans un domaine donné peuvent être gérés sur un seul périphérique).

L'*horloge* définit le type de périphérique sur lequel l'architecture peut fonctionner. Kerberos utilise l'heure réelle (timestamps) même sur le serveur de ressources, une synchronisation du temps entre les entités est requise.

Le *service cache* indique les architectures qui fournissent un cache pour les données, au-delà du potentiel serveur mandataire permis grâce à l'usage de CoAP.

La *taille des messages* durant la communication est un facteur important ; dans le réseau considéré (IEEE 802.15.4), la taille maximale de la charge utile d'un paquet est fortement limitée. Kerberos utilise le format ASN.1 (BER) ainsi qu'un nombre important d'informations, certaines avec des tailles arbitraires (des noms de domaine par exemple), ce qui amène à des messages à la connexion qui ont une taille dépassant largement le MTU du lien IEEE 802.15.4. Les architectures basées sur DTLS prennent 29 octets de charge utile pour la couche de sécurité, 8 de moins quand TLS est utilisé (sans prendre en compte TCP). Les architectures basées sur COSE permettent aux applications d'échanger des messages sécurisés qui se suffisent à eux-mêmes, sans connexion, et donc un message sécurisé et signé avec ECDHE aurait une grande taille. Cependant, ces architectures peuvent également réutiliser des associations de sécurité, ce qui permettrait de ne pas avoir à effectuer un échange ECDHE à chaque communication.

Dans nos exemples, 6LoWPAN est utilisé pour la couche réseau. La taille effective des messages devrait être nuancée au regard de l'optimisation offerte par 6LoWPAN-GHC.

Le *nombre de messages* est une valeur importante pour les périphériques sur batterie. Les connexions persistantes entre les entités dans une architecture ne sont pas comptées puisqu'elles n'ont qu'une seule occurrence, lorsque l'appareil démarre.

Le nombre de messages échangés dépend :

1. du nombre de clients et de la fréquence de leurs demandes s'il n'y a pas de cache ;
2. de la fréquence des mises à jour s'il y a un cache et que les échanges sont réduits aux mises à jour du cache de la représentation des ressources.

Note sur le nombre de messages : *nombre de messages à la première connexion (SR)* inclut seulement les messages de la connexion, pas la requête ni sa réponse. Une connexion DTLS implique au moins 10 messages (sans certificats) et jusqu'à 14 messages avant qu'une donnée ne soit transférée. Quand un serveur de ressources est supposé échanger des messages avec un service d'authentification ou d'autorisation, ce canal de communication est considéré déjà ouvert (pas de connexion DTLS), c'est pour cela que l'introspection dans l'architecture ACE OAuth n'ajoute que 2 messages.

5.3.3.3 Aspects généraux

Le critère *architectures avec application spécifique* signifie que l'architecture est liée à un protocole applicatif. C'est le cas de MQTT-SN qui définit son propre protocole applicatif par exemple.

Finalement, le mécanisme de *découverte de ressources* fourni directement par l'architecture permet à un client de gagner du temps et de l'énergie pour trouver des ressources. 6LoWPAN associé au protocole CoAP permet à un client cette découverte de ressources, mais cela requiert de la communication de groupe (via le multicast) ce qui n'est pour le moment pas sécurisé, et peut ne pas être permis dans le réseau.

5.4 Conclusion

Ce chapitre a utilisé la définition d'une architecture de sécurité afin de trouver des architectures présentes dans la littérature. Nous avons décrit ces architectures selon notre méthode illustrée au chapitre précédent. Nous avons ensuite établi deux scénarii pour donner un contexte à la comparaison ; le client est soit dans le réseau de capteurs soit sur Internet. Les critères de comparaison ont été présentés, ils sont nombreux et illustrent les points d'intérêt pour le choix d'une architecture ; ces critères se basent sur les contraintes des matériels, les algorithmes utilisé (cryptographie symétique ou asymétrique), la granularité de l'autorisation, le nombre de messages échangés entre les domaines ou encore la taille des en-têtes.

Le chapitre suivant est dédié à l'interprétation des résultats présentés ici sous la forme d'un tableau comparatif. Nous montrerons les avantages et les inconvénients de chaque architecture suivant différentes contraintes de déploiement (contraintes matérielles, nombre de clients, fréquence des requêtes, etc.).

Chapitre 6

Guide des architectures

Nous décrivons dans ce chapitre les architectures pertinentes selon des contraintes de déploiement. Par exemple, une architecture peut être efficace pour un déploiement avec un grand nombre de clients parce qu'un serveur mandataire (ou un cache) non contraint est en charge de la distribution des messages, mais inefficace d'un point de vue de la latence car chaque message est signé en utilisant un algorithme asymétrique. Les recommandations suivantes sont basées sur des tableaux comparatifs expliqués dans le chapitre précédent.

Ce chapitre présente les contributions suivantes :

1. Des contraintes de déploiement et des contraintes sur les nœuds ;
2. Le classement des architectures face à ces contraintes.

6.1 Architectures non adaptées aux environnements contraints

Dans cette section, nous présentons et expliquons pourquoi certaines architectures ne sont pas adaptées aux environnements contraints.

Notre analyse est centrée sur les réseaux avec des fortes limitations, d'une part sur le MTU (tel que dans IEEE 802.15.4 avec un maximum de 127 octets), et d'autre part sur l'alimentation électrique (machines potentiellement alimentées par batterie). Par conséquent, le nombre de messages transmis doit être minimisé. Nous nous concentrons sur du matériel peu onéreux pour les serveurs de ressources, ce qui implique de limiter les calculs sur ce matériel et de consommer le minimum de mémoire possible.

6.1.1 Kerberos

Les problèmes de Kerberos dans les environnements contraints sont nombreux. Premièrement, nous pouvons citer la taille des messages [1], au format ASN.1, qui implique d'ajouter des octets à chaque information encodée. Par exemple, une simple constante sur un octet prend 5 octets une fois encodée, et 19 octets pour un temps [2]. Ce problème de longueur de message se retrouve dès l'authentification, avec une taille qui dépasse le MTU. Pour pallier à ce problème, les messages doivent être fragmentés, soit au niveau IP, soit au niveau du protocole de transport, comme TCP (et prenant de fait 20 octets par message). La taille des messages pose également problème durant la communication entre le client et le serveur de ressources, Kerberos utilise au moins 72 octets de charge utile. La majorité du message ne sert qu'à protéger la donnée, et la fragmentation est nécessaire, laissant seulement 4 octets pour la

1. Même sur des réseaux classiques tels que Ethernet, Kerberos a un problème de MTU et TCP est utilisé pour surmonter ce problème.

2. Le format choisi est lisible par un humain, donc très inefficace en mémoire, en temps de calcul, en taille de code ainsi que par rapport au MTU.

donnée[3]. Le format de message n'est pas adapté aux environnements contraints avec un MTU aussi limité

Deuxièmement, le protocole Kerberos nécessite des horloges sur le client et le serveur de ressources, elles sont utilisées durant l'authentification et pour vérifier la validité des requêtes (les autorisations étant limitées dans le temps). Les horloges temps-réel ne sont pas toujours disponibles sur du matériel très contraint, et lorsqu'elles sont disponibles, elles ne sont pas toujours de bonne qualité (il faut les resynchroniser régulièrement). Par conséquent, les serveurs de ressources doivent synchroniser leur horloge pour gérer un décalage.

Troisièmement, aucune authentification basée sur les certificats n'est proposée par Kerberos. L'entité d'authentification est séparée du serveur de ressources, l'authentification peut alors être gérée par un matériel non contraint, il serait donc possible de gérer l'authentification avec des certificats. Actuellement, une communication doit être protégée par un mot de passe ce qui est fastidieux dans le premier scénario, puisque l'appareil n'est pas accessible à l'autorité

Quatrièmement, le protocole Kerberos nécessite de la mémoire : un contexte est conservé par client sur le serveur de ressources, qui doit stocker différentes informations requises pour l'authentification. Les informations stockées sont, entre autres, des noms de domaine ou des noms arbitraires (comme le nom du royaume ou de l'autorité) ou bien différents temps (comme la date de la requête), etc.

En conclusion, l'architecture Kerberos telle que originellement spécifiée n'est pas compatible avec un MTU limité. Kerberos nécessite des horloges temps-réel, une synchronisation régulière avec les serveurs de ressources, beaucoup d'informations, et n'est donc pas adapté au premier scénario. Le seul avantage de Kerberos comparé aux autres architecture est la séparation des entités d'authentification et d'autorisation, mais qui n'est requis par aucun de nos scénarii. Cette architecture est donc non adaptée aux environnements contraints.

6.1.2 SSH

SSH est un ensemble très répandu de protocoles, de conception simple, utile pour beaucoup d'applications mais pas adapté aux environnements contraints.

En effet, la connexion nécessite l'usage d'un algorithme asymétrique pour l'authentification et l'échange de clés (peu importe le scénario). L'authentification implique alors de longs calculs sur le serveur de ressources à l'établissement de la connexion. Le calcul dure plusieurs dizaines de secondes pour le partage de la clé sur un microcontrôleur 8 MHz [82], même en utilisant de la cryptographie à base de courbes elliptiques, avec une taille de clés considérée insuffisante par le NIST[4]. La cryptographie à clé publique obligatoire implique l'usage de matériel spécialisé pour les nœuds contraints, et donc influe sur le coût de déploiement.

De plus, l'architecture nécessite une couche de transport TCP, contrairement aux autres architectures qui peuvent être utilisées avec UDP. L'usage de TCP implique un surcoût au niveau de la taille des messages, TCP ayant un en-tête de 20 octets, et l'augmentation de la taille des messages peut également impliquer plus de messages lors de la poignée de mains.

Enfin, l'architecture SSH, comme les autres architectures basées sur un protocole, manque d'une gestion centralisée de l'authentification. Les serveurs de ressources sont obligés de stocker des informations à propos de chaque client. Par conséquent, le nombre de clients et de connexions simultanées est limité par la mémoire disponible sur les serveurs de ressources de l'architecture.

En conclusion, l'architecture SSH ne se démarque pas des autres architectures, peu importe la métrique considérée : la taille ou le nombre de messages, la cryptographie utilisée, les fonctionnalités requises sur les serveurs de ressources ou encore la mémoire utilisée.

3. Dans un environnement 802.15.4, une architecture Kerberos pourrait utiliser 23 octets pour l'en-tête de couche 2, 20 octets pour 6LoWPAN, 4 octets pour UDP, 72 octets pour Kerberos, et enfin 4 octets pour CoAP. Il ne reste donc bien que 4 octets pour la charge utile, à condition de ne pas utiliser TCP ni de chiffrement au niveau de la couche liaison.

4. Une taille de clé de 160 bits, telle que utilisée dans [82] est équivalente à une clé RSA de 1024 bits [28], ce qui est obsolète. De plus, l'augmentation du temps de calcul n'est pas linéaire par rapport à l'augmentation de la taille de la clé.

6.1.3 IPsec

IPsec est conçu pour protéger différentes communications (en les discriminant sur leurs adresses IP, les ports ou le protocole) avec différents secrets et algorithmes. Ce protocole est pensé pour les communications sur Internet pour protéger diverses communications entre des nœuds distants. Dans les environnements contraints, chaque serveur de ressources a généralement une seule tâche, donc la discrimination ne se passe que sur les adresses IP, ce qui est inutile dans notre cas : une seule association de sécurité est suffisante entre le client et le serveur de ressources qui n'a qu'une seule tâche. Finalement, les fonctionnalités spécifiques d'IPsec ne sont pas utiles dans les environnements contraints tels que nous les avons décrits dans ce document, son usage reste pertinent entre des routeurs, mais cela est hors du cadre de notre étude.

6.1.4 Conclusion

Kerberos a un problème de taille de messages, SSH nécessite des algorithmes asymétriques, et les fonctionnalités d'IPsec ne sont pas pertinentes dans les environnements contraints. Aucune de ces trois architectures n'est donc à recommander pour des environnements contraints.

6.2 Contraintes de déploiement

Dans cette section, nous détaillons des contraintes de déploiement (nombre de clients, fréquence des demandes, etc.) et classons les architectures selon ces contraintes.

Chaque déploiement a un objectif particulier, mais tous les déploiements partagent des paramètres communs (et potentiellement des contraintes). Parmi ces paramètres, nous retrouvons le nombre de clients, la fréquence de leurs requêtes, la granularité de l'autorisation offerte par l'architecture, etc. Dans cette section, nous ordonnons les architectures selon leur pertinence par rapport à ces paramètres.

6.2.1 Grand nombre de clients

Pour les déploiements avec un grand nombre de clients, les architectures avec un serveur mandataire (MQTT-SN et OSCAR) ont un avantage puisque le nombre de clients et le nombre de messages transmis aux serveurs de ressources ne sont pas corrélés (avec un cache par exemple). De plus, le serveur mandataire peut également effectuer l'authentification et l'autorisation, et donc les serveurs de ressources n'ont pas à stocker d'informations à propos de chaque client, réduisant leur consommation de mémoire.

6.2.2 Haute fréquence de demandes

Pour les déploiements avec une haute fréquence de requêtes (avec des données rapidement obsolètes), disposer d'un cache n'est pas suffisant : la latence des requêtes devient une métrique importante. Les architectures les plus adaptées dans ce contexte sont celles récoltant les données dans un cache avant d'être demandées par les clients (OSCAR, MQTT-SN) : il n'y a pas d'authentification entre les clients et les serveurs de ressources (moins de messages générés, moins de contrainte sur la mémoire des serveurs de ressources, moins de connexions à maintenir). Les architectures sans cache sont moins efficaces. Les paramètres tels que l'usage de cryptographie asymétrique et le nombre de relations entre les domaines sont à prendre en compte pour choisir une architecture. Selon ces critères, voici le classement des architectures de la plus à la moins adaptée : ACE DCAF (pas de cryptographie asymétrique), ACE OAuth (pas de cryptographie asymétrique sauf si DTLS est utilisé pour contacter SR), puis OSCORE (requiert de la cryptographie asymétrique mais seulement deux messages sont requis par requête) et finalement A-DTLS qui nécessite de la cryptographie asymétrique.

6.2.3 Déploiements sans serveur mandataire

Dans notre document, nous considérons un serveur mandataire comme non contraint, ainsi il peut être utilisé comme un cache de données.

Pour les déploiements avec des nœuds indépendants sans serveur mandataire (e.g. : une ampoule connectée), la mémoire disponible sur les serveurs de ressources est un problème. Dans ce cas, les serveurs de ressources doivent (i) mémoriser l'identité et le matériel cryptographique de chaque client, et (ii) ils doivent gérer plusieurs connexions simultanées avec des informations qui leur sont liées (par exemple le matériel cryptographique, l'état ou des temporisateurs).

OSCAR fournit une atténuation simple pour (i) et (ii) : l'architecture nécessite de stocker, sur des serveurs de ressources, une clé pour chaque type de ressources uniquement et non une pour chaque client. Chaque architecture peut supprimer l'association de sécurité après chaque requête, permettant aux serveurs de ressources de stocker peu de contextes (algorithmes et clés négociés, options, numéros de séquence et ainsi de suite), cela est la méthode la plus efficace pour atténuer (ii). Malheureusement, OSCAR nécessite une entité d'authentification pour gérer des clients et diminuer le travail à faire sur le serveur de ressources. De ce fait, la seule manière pour une architecture de se démarquer des autres est d'avoir moins de messages transmis entre les domaines (en incluant la communication avec l'entité d'authentification).

Dans les déploiements avec des serveurs de ressources indépendants (i.e. sans serveur central d'authentification ni d'autorisation), la communication peut être réduite simplement au protocole de communication sécurisé. La plupart des architectures utilisent DTLS pour sécuriser les échanges entre les domaines : OSCAR, ACE DCAF, ACE OAuth, MQTT-SN, A-OSCORE (au moins dans le premier scénario) et l'architecture éponyme. A-DTLS n'utilise que les protocoles DTLS et CoAP entre deux appareils, les autres architectures échangent plus de messages car elles utilisent des appareils ou des protocoles supplémentaires. Si le critère de sélection de l'architecture est basé entièrement sur le nombre de messages échangés, alors les architectures restantes sont A-DTLS et A-IPsec. A-OSCORE peut être utilisé mais uniquement avec des clés pré-configurées.

6.2.4 Granularité d'autorisation attendue

La granularité de l'autorisation varie selon l'architecture. ACE DCAF, ACE OAuth et MQTT-SN ont une granularité d'autorisation jusqu'à la ressource. OSCAR accorde des permissions aux clients avec une granularité jusqu'au type de ressources sur tous les serveurs de ressources à la fois, mais ce système implique une clé unique partagée sur plusieurs appareils : un seul serveur de ressources avec une vulnérabilité peut affecter la sécurité de tous les serveurs de ressources. Les architectures basées sur un seul protocole (DTLS, IPsec et OSCORE) délèguent l'autorisation à l'application, la granularité peut être jusqu'à la ressource mais doit être configurée sur chaque serveur de ressources ce qui peut être contraignant pour de grands déploiements.

6.2.5 Déploiements avec actionneurs

Demander une donnée à un serveur de ressources n'est pas différent d'envoyer une commande à un actionneur pour toutes les architectures, excepté MQTT-SN qui n'utilise pas CoAP. Dans l'architecture MQTT-SN, les clients ne communiquent pas avec les serveurs de ressources, ils ne leur envoient aucune information directement. MQTT-SN est conçu comme un système de publication et de souscription, les clients demandent à recevoir des mises à jour d'un sujet (une ressource). Par conséquent, cette architecture est adaptée à la récolte de données, mais peu adaptée aux déploiements avec des actionneurs [5].

5. Dans l'architecture MQTT-SN, nous pourrions faire fonctionner des actionneurs au prix d'un détournement du fonctionnement initial. Par exemple, le serveur de ressources avec des actionneurs s'inscrit à un sujet auprès du serveur mandataire, et le client s'enregistre comme diffuseur sur ce sujet. De ce fait, quand le client va publier un message, il sera retransmis au serveur de ressources, simulant une requête. Ce détournement du protocole implique plus de messages de contrôle.

6.2.6 Déploiements avec un réseau hétérogène

Les appareils communiquant entre eux doivent utiliser les mêmes algorithmes de cryptographie : ils peuvent négocier ces algorithmes à la volée ou avoir un accord préalable par exemple. Toute architecture utilisant un protocole de négociation (DTLS, IPsec) est compatible avec des réseaux hétérogènes (les architectures éponymes, ainsi que ACE DCAF, MQTT-SN). OSCAR et OSCORE nécessitent un accord préalable, ce qui peut être contraignant avec de grands réseaux d'appareils communiquant entre eux. ACE OAuth nécessite également un accord préalable entre les clients et l'entité d'authentification sauf si DTLS ou EDHOC sont utilisés.

Mise à part la perspective de sécurité ou de contraintes sur les appareils, MQTT-SN n'a pas de négociation de la représentation du type de ressources (contrairement à CoAP). De ce fait, MQTT-SN n'est pas adapté aux déploiements hétérogènes, avec différentes représentations d'un même type de ressources.

6.3 Contraintes sur les nœuds

Dans cette section, nous détaillons des contraintes sur les nœuds (possession d'une horloge, mémoire disponible, etc.) et classons les architectures selon ces contraintes.

Dans les environnements contraints, les appareils ont peu de capacités. Dans la suite, nous présentons quelques recommandations selon les limitations des appareils.

6.3.1 Horloges

Dans le premier scénario, quand les clients sont hors du domaine serveur, les certificats peuvent être utilisés pour limiter la période de validité de la clé. Cela implique l'usage d'une horloge sur le serveur de ressources sauf s'il existe une entité d'authentification sur un autre appareil (un serveur mandataire). OSCAR est la seule architecture nécessitant l'usage de certificats et donc d'horloges sur chaque appareil. Toute architecture pouvant, pour les deux scénarii, utiliser une authentification par clé pré-configurée est compatible avec un déploiement sans horloge.

6.3.2 Mémoire

Les appareils déployés peuvent avoir moins de 10 kilooctets de mémoire RAM et moins de 100 kilooctets de mémoire Flash. Quelques paramètres consomment de la mémoire comme les protocoles et les algorithmes utilisés, les méthodes d'authentification des clients et les informations de session.

	Contexte par client	Protocole Applicatif	Protocole de Sécurité	Cryptographie Asymétrique	Autorisation
OSCAR	non	CoAP	DTLS	**oui** (et certificats)	oui
ACE DCAF	**oui** (une fois connecté)	CoAP	DTLS	non	oui
ACE OAuth	**oui** (DTLS ou COSE)	CoAP	DTLS (optionnel), OSCORE	optionnel	oui
MQTT-SN	non	MQTT-SN	DTLS	optionnel	non
A-DTLS	**oui** (sans cache)	CoAP	DTLS	optionnel	non
A-IPsec	**oui** (sans cache)	CoAP	IPsec	optionnel	non
A-OSCORE	**oui** (sans cache)	CoAP	DTLS (optionnel), COSE	optionnel	non

TABLE 6.1 – Mémoire utilisée : algorithmes et protocoles requis par les architectures.

Le tableau 6.1 présente un aperçu de la consommation mémoire des architectures. Seuls OSCAR et MQTT-SN ne nécessitent pas de stocker des informations concernant chaque client sur les serveurs de ressources. Les architectures utilisent CoAP comme protocole applicatif, excepté MQTT-SN qui utilise un protocole plus simple (pas de gestion des types de message, peu de métadonnées). Les architectures utilisent DTLS comme protocole de négociation et de sécurité excepté IPsec. OSCORE et ACE OAuth peuvent éviter l'usage de DTLS si un accord préalable a été fait. ACE OAuth et OSCORE nécessitent les procédures COSE, ACE OAuth implémente OSCORE, et donc par définition nécessite plus de

mémoire. OSCAR est la seule architecture nécessitant des algorithmes de cryptographie asymétrique. Finalement, ACE DCAF, ACE OAuth et OSCAR nécessitent des procédures d'autorisation.

MQTT-SN est l'architecture la moins gourmande en mémoire, elle nécessite DTLS et le protocole MQTT-SN, sans avoir à stocker de contexte par client. ACE DCAF et ACE OAuth ont un serveur d'authentification (qui peut être centralisé en un appareil unique pour tout le réseau) pour stocker les informations des clients. Ces architectures nécessitent uniquement que le serveur de ressources stocke des informations de session pour les connexions établies, ce sont les deuxièmes architectures les moins consommatrices de mémoire. Dans les déploiements avec un grand nombre de clients, OSCAR se démarque dans les architectures restantes puisqu'il possède un serveur cache. Finalement, les autres architectures ont une consommation de mémoire similaire [6].

6.3.3 Calculs et batteries

Les calculs lourds, comme les algorithmes asymétriques, induisent un épuisement des batteries et devraient donc être évités. Cependant un compromis entre calcul et mémoire peut se faire : quand la cryptographie asymétrique est uniquement utilisée pour la création du canal de communication (échange de clé ou authentification), le canal peut rester ouvert.

MQTT-SN ne nécessite pas d'algorithme asymétrique pour les serveurs de ressources. Cependant, MQTT-SN est inefficace quand il n'y a pas de client pour une ressource puisque le serveur de ressources continue d'envoyer des mises à jour. La fréquence de mise à jour dépend des changements de la ressource et de la configuration du capteur.

DTLS, IPsec et OSCORE nécessitent des algorithmes asymétriques mais uniquement pour l'authentification et dans le premier scénario (clients étrangers au domaine serveur).

OSCAR nécessite des algorithmes asymétriques : une ressource qui change implique que le serveur de ressources signe la nouvelle valeur, ce qui coûte du temps et de l'énergie et aucun compromis ne peut être fait.

6.4 Résumé

	plus favorable	adapté	moins adapté
Grand nombre de clients	toute architecture avec un serveur mandataire MQTT-SN, OSCAR	ACE DCAF, ACE OAuth	architectures sans serveur mandataire A-DTLS, A-IPsec, A-OSCORE
Haute fréquence de requête	toute architecture avec un serveur mandataire MQTT-SN, OSCAR	ACE DCAF, ACE OAuth	architectures sans serveur mandataire A-DTLS, A-IPsec, A-OSCORE
Réseaux sans serveur dédié à l'authentification	A-DTLS, A-IPsec	A-OSCORE (si accord préalable)	toutes les autres architectures
Granularité d'autorisation	MQTT-SN, ACE DCAF, ACE OAuth	OSCAR	A-DTLS, A-IPsec, A-OSCORE
Réseaux avec actionneurs	-	toutes sauf MQTT-SN	MQTT-SN
Réseaux hétérogènes	architectures avec protocole de négociation ACE DCAF, A-DTLS, A-IPsec	-	MQTT-SN (représentation de ressources non négociable)
Horloge temps-réel	MQTT-SN, ACE DCAF, ACE OAuth	toutes sauf OSCAR	OSCAR
Mémoire	MQTT-SN, ACE DCAF, ACE OAuth	OSCAR	architectures sans serveur dédié pour l'authentification : A-DTLS, A-IPsec, A-OSCORE (si grand nombre de clients)
Calculs et batteries	MQTT-SN, ACE DCAF, ACE OAuth	A-DTLS, A-IPsec, A-OSCORE	OSCAR

Table 6.2 – Résumé de la compatibilité des architectures selon la contrainte considérée

6. CoAP permet aux architectures d'avoir un serveur cache, dans ce cas le point de connexion peut se terminer avant les serveurs de ressources, réduisant le poids des communications et l'empreinte mémoire.

Le tableau 6.2 présente un résumé des précédentes comparaisons. Aucune des architectures ne se démarque dans toutes les situations. Les architectures avec un serveur mandataire qui authentifie les clients, et met en cache les réponses des serveurs de ressources, tendent à être adaptées aux contraintes des déploiements. Selon l'architecture, le serveur mandataire cache la complexité de la gestion des clients aux serveurs de ressources. La donnée est alors mise en cache et les algorithmes de sécurité sont différents sur les clients et les serveurs de ressources.

6.5 Remarques

Aucune architecture ne fournit un mécanisme intégré pour la découverte de ressources, cela doit se faire avec une combinaison de 6LoWPAN et de CoAP (et DTLS pour une découverte sécurisée).

Le principe de séparation du processus d'authentification utilisé dans Kerberos est utile pour construire un système d'authentification Single Sign On (SSO) pour les réseaux d'entreprise, avec de multiples fonctionnalités accessibles aux employés depuis leur ordinateur de bureau (comme un service d'impression par exemple). Même si Kerberos n'est pas adapté aux environnements contraints, il peut être utilisé dans un autre contexte, comme dans les déploiements prenant en charge la gestion de la vie privée. Par exemple, nous pouvons créer une architecture qui réponde à des clients sans connaître leur identité, en partant des principes suivants :

1. un domaine A fait confiance à une entité d'authentification d'un domaine B ;
2. cette entité fournit au client un ticket qui atteste qu'il a bien été authentifié mais ne divulgue pas son identité ;
3. une entité d'autorisation du domaine A accomplit sa tâche, en utilisant seulement le ticket et sans savoir quelle est l'identité du client.

De cette manière, l'identité du client est connue de l'entité d'authentification du domaine B, mais ne l'est pas des autres domaines[7]. Comme exemple pratique, une entreprise d'impression peut autoriser l'accès à toute personne authentifiée par un serveur d'authentification d'une entreprise associée, sans identifier la personne lançant l'impression. L'architecture ACE DCAF permettrait également, avec ses serveurs d'authentification, de protéger l'identité du client.

6.6 Conclusion

Dans les chapitres précédents, nous avons décrit ce qu'est une architecture de sécurité et nous avons présenté une vue d'ensemble des principales architectures disponibles dans la littérature (fonctionnement, contraintes sur les nœuds). Dans ce chapitre, nous avons défini un guide qui permet de choisir une architecture logicielle selon ses besoins, dictés par des contraintes de déploiement ou de matériel.

Pour développer ce guide, nous avons fait une comparaison selon deux scénarii avec différents déploiements contraints. Cette comparaison montre qu'un serveur mandataire (ou au moins un service d'authentification indépendant du serveur de ressources) est un composant clé pour concevoir des architectures efficaces. Les architectures avec un serveur mandataire et un service de cache sont davantage adaptées aux contraintes de déploiement. L'architecture maître-esclave MQTT-SN est la mieux adaptée aux contraintes de déploiement et sur les nœuds, sa conception permet une séparation nette entre le réseau contraint et Internet : les nœuds contraints ne sont pas en liaison avec les clients, ils sont donc moins sollicités et nécessitent moins de mémoire. Cette architecture maître-esclave a cependant des défauts qui limitent son déploiement aux réseaux sans actionneurs et ne permettent pas l'hétérogénéité des matériels (représentation des ressources non négociables).

7. Des informations utiles peuvent également être ajoutées au ticket par le tiers de confiance pour un usage futur : les clients peuvent être catégorisés dans des groupes, et cette information est partagée entre le tiers de confiance (l'entité d'authentification) et l'entité d'autorisation via le ticket.

Dans le chapitre suivant, nous présenterons une amélioration du mécanisme de sécurité DTLS. Ce mécanisme étant utilisé par la plupart des architectures étudiées, son amélioration bénéficie à ces architectures.

Chapitre 7

Optimisation de protocole de sécurité

Dans les chapitres précédents, nous avons vu que le coût d'une architecture se chiffre de multiples manières : nombre et taille des paquets échangés, algorithmes utilisés, mémoire requise, etc. Ces paramètres sont liés à la répartition des fonctions cryptographiques sur les nœuds (authentification, autorisation) ainsi qu'aux protocoles de communication utilisés. Un protocole revient souvent dans les architectures observées : DTLS. L'IETF promeut l'usage de DTLS dans les environnements contraints tels que l'Internet des objets, et il est utilisé dans les architectures OSCAR, ACE DCAF, ACE OAuth, MQTT-SN, et enfin A-DTLS. Une manière d'améliorer les architectures serait d'optimiser le protocole de communication sécurisée. Dans ce chapitre, nous proposons des améliorations du protocole DTLS afin de l'adapter à des contraintes réseau. L'objectif est d'améliorer une architecture de référence, mesurer cette amélioration puis comparer avec les avantages d'une architecture maître-esclave.

Ce chapitre présente les contributions suivantes :

1. L'optimisation du protocole de sécurité DTLS à travers la réduction du nombre de messages et de la taille des en-têtes ;
2. La validation de ces optimisations par expérimentation.

7.1 Introduction

Malgré l'utilisation d'UDP comme protocole de transport, DTLS ajoute un coût à la connexion pour deux raisons. D'une part, le nombre de messages transmis durant la poignée de mains retarde l'échange de données applicatives, ce qui est d'autant plus important dans les réseaux avec rapport cyclique [83], et d'autre part, l'en-tête de DTLS est importante, et laisse peu de place aux données applicatives dans les paquets.

Ce chapitre présente deux améliorations du protocole DTLS, d'une part en matière de vitesse de connexion, en réduisant le nombre de messages de signalisation, et d'autre part en taille de charge utile, via la réduction de la taille des en-têtes en s'appuyant sur les recommandations de l'IETF. Afin de centrer les travaux sur la couche DTLS uniquement, nous ne ferons aucune supposition sur les autres couches utilisées.

7.2 Objectif et choix

DTLS sécurise une connexion entre deux entités. Le protocole est composé d'une négociation de matériel et d'algorithmes cryptographiques, résumée dans le tableau 5.7 du chapitre 5 (page 55), et d'un échange de données chiffrées, authentifiées et dont l'intégrité est vérifiée. DTLS utilise UDP et non TCP comme protocole de transport, le protocole de plus haut niveau doit, si besoin, s'occuper de la gestion des retransmissions de paquets perdus, de l'ordonnancement et de la fragmentation.

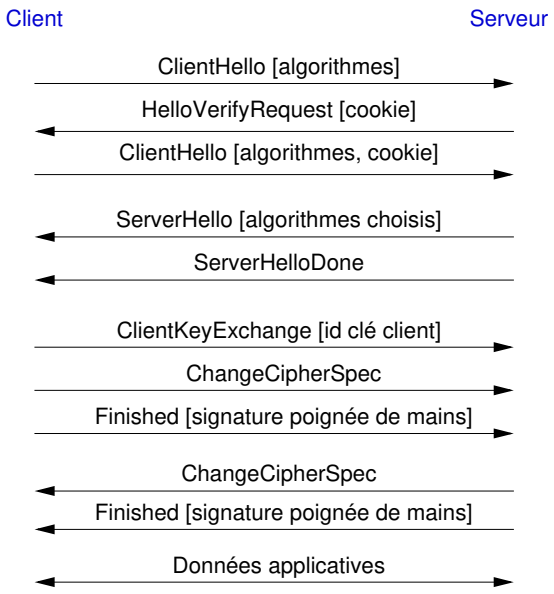


FIGURE 7.1A – Poignée de mains DTLS. Les messages optionnels ont été supprimés. La charge utile des messages est entre crochets.

Nous avons deux objectifs. Premièrement, nous souhaitons établir une connexion aussi rapidement que possible, avec des matériels très contraints sur un réseau contraint de type IEEE 802.15.4. Nous réduisons donc le nombre de messages pour diminuer le nombre d'aller-retours, le temps d'attente sur le réseau et le temps de transmission. Deuxièmement, nous souhaitons augmenter la charge utile des messages, donc nous diminuons la taille des en-têtes DTLS.

Le choix des algorithmes est basé sur les recommandations du groupe travail DICE de l'IETF [84], qui a défini un profil d'utilisation de DTLS pour les environnements contraints. Dans ces recommandations, nous trouvons un chiffrement CCM avec 8 octets de MAC, et un algorithme de chiffrement symétrique AES avec des clés de 128 bits. Pour le partage des clés, le groupe DICE recommande l'usage d'un secret partagé, et, si cela n'est pas envisageable, un échange Diffie-Hellman ainsi qu'une authentification par certificats avec des courbes elliptiques (ECDSA). Nous sélectionnons l'usage d'un secret partagé et d'algorithmes symétriques uniquement, ce qui correspond au scénario 2 du chapitre 5. Les choix faits pour les environnement contraints seront toujours valides pour les environnements moins contraints.

L'optimisation de la poignée de mains que nous proposons est valable pour les communications avec authentification par secret partagé. L'optimisation de la taille des paquets est pertinente pour toute communication DTLS. Néanmoins, ces optimisations ne sont pas compatibles avec le standard.

7.3 Optimisations de DTLS

Dans cette section, nous présentons nos deux optimisations : une poignée de mains plus rapide pour réduire le délai avant l'envoi de la première donnée et diminuer ainsi l'empreinte d'une connexion sur le réseau, et la réduction de la taille des en-têtes lors des échanges pour réduire la fragmentation et la durée des échanges.

7.3.1 Poignée de mains plus rapide

DTLS est un protocole verbeux qui n'a pas été conçu pour les réseaux très contraints, tel que présenté à la section 3.1.11 page 22. La figure 7.1b montre la poignée de mains optimisée, où le

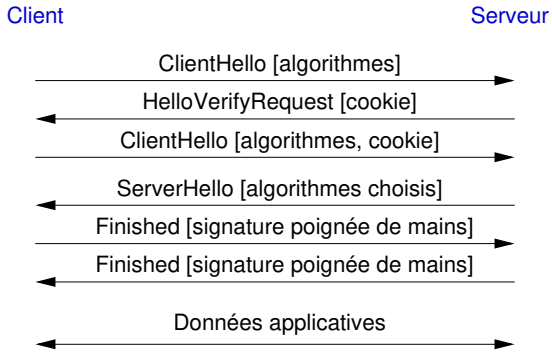


FIGURE 7.1B – Poignée de mains optimisée de DTLS.

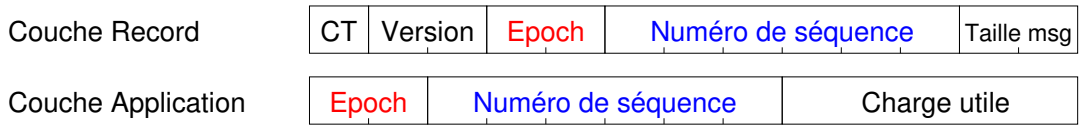


FIGURE 7.2 – Redondance dans les en-têtes, lié à un usage conventionnel de vecteur d'initialisation dans les algorithmes de chiffrement AEAD.

nombre de paquets échangés est réduit. Comme nous utilisons des clés préchargées, un échange Diffie-Hellman n'est donc pas nécessaire. Nous n'utilisons pas non plus de certificats, afin de ne pas utiliser de cryptographie asymétrique, donc les messages optionnels, requis par le système de certificats, sont omis. Nous supprimons les messages servant uniquement de signalisation et qui ne contiennent aucune donnée utile (entre crochets) : **ServerHelloDone** et **ChangeCipherSpec**. Ces messages ne sont plus utiles car nous avons dorénavant une séquence fixe de messages, l'indication apportée par un message de signalisation n'est plus nécessaire. Nous supprimons les messages contenant l'identité des pairs ou de la clé à utiliser : **ClientKeyExchange** et **ServerKeyExchange**. L'identification du client peut se faire de deux manières : soit le serveur utilise l'adresse IP comme identifiant du client (ou l'adresse MAC s'ils sont dans le même réseau), soit le protocole applicatif est utilisé pour communiquer l'identité du client au serveur. Concernant l'identification de la clé, nous considérons que nous n'utilisons qu'une seule clé entre deux appareils.

L'adresse MAC (ou IP) est utilisée pour identifier un pair, mais un attaquant ne peut en retirer une information utile. L'attaquant peut usurper une adresse mais pas la connexion s'il ne connaît pas le secret partagé, qui n'est jamais échangé en clair sur le réseau.

Avec ces optimisations, nous retirons jusqu'à 5 messages durant la poignée de mains et fixons le nombre de messages échangés (plus de message optionnel). Nous avons retiré des messages sans ajouter de calcul supplémentaire, par conséquent la poignée de mains sera plus rapide. Cela simplifie le code final, et il est attendu que le binaire ait une taille réduite, moins d'erreurs d'implémentation, moins de maintenance et une lisibilité améliorée. L'inconvénient de ces optimisations est qu'elles rendent notre solution incompatible avec les implémentations de DTLS.

7.3.2 Charge utile plus importante

Une fois que la connexion est établie, DTLS transmet les données applicatives utilisant deux couches (Record et Application) dont les en-têtes sont montrés en figure 7.2. La partie Record contient deux en-têtes (Epoch et Sequence Number) utilisés pour détecter les pertes de paquets, les duplications et les arrivées non ordonnées. Quand un algorithme AEAD est utilisé, la partie Application Data contient à la fois la charge utile et un vecteur d'initialisation. Le vecteur d'initialisation doit être

unique pour chaque message, et il est composé conventionnellement des champs Epoch et Sequence Number de la couche Record.

Nous pouvons supprimer cette redondance (8 octets) dans les deux couches, réduisant la surcharge protocole de DTLS de 29 à 21 octets. Cela représente un gain de 6 % de charge utile totale d'un paquet sur un réseau IEEE 802.15.4.

Le gain est surtout lié à la taille maximale d'un paquet (97 octets au lieu de 89 dans un réseau IEEE 802.15.4, soit 8,2 %). Les applications ont plus de place pour leurs données dans un seul envoi. De plus, il est attendu que la suppression de cette redondance améliore les durées de transmission.

L'inconvénient de cette optimisation, comme les précédentes, est qu'elle rend notre solution incompatible avec les implémentations de DTLS.

7.4 Environnement de test

Dans cette section, nous présentons l'environnement matériel et la pile de protocoles utilisée pour la validation des optimisations lors de l'expérimentation.

Nous faisons une expérimentation pour mesurer les améliorations apportées par notre solution en termes de consommation mémoire (RAM et Flash), et de temps de connexion (lors de la poignée de mains et de l'échange de messages).

Dans cette section, nous décrivons l'environnement matériel et logiciel de notre expérience.

7.4.1 Plateforme matérielle

La plateforme matérielle est basée sur un micro-contrôleur ATMega128RFA1 (Zigduino) avec une connectivité IEEE 802.15.4 native, 16 Ko RAM, 128 Ko Flash et une fréquence de 16 MHz. La connectivité 802.15.4 est en mode pair-à-pair, la radio est toujours allumée pour éviter des délais de réveil trop longs de la couche liaison. Il n'y a qu'un saut du client au serveur.

Le premier appareil agit comme un serveur DTLS qui écoute sur le réseau, le second agit comme un client DTLS qui envoie périodiquement un message avec une taille arbitraire.

7.4.2 IEEE 802.15.4 et CSMA-CA

Le temps d'attente 802.15.4 de l'algorithme CSMA-CA est de 1,12 ms en moyenne, voir chapitre 2.

L'en-tête de la couche liaison utilise au moins 9 octets, mais nous en utilisons 11 parce que nous ajoutons le Personal Address Network (PAN) en plus des quatre octets d'adressage MAC (source et destination), afin de pouvoir lancer plusieurs expérimentations dans notre laboratoire dans des sous-réseaux séparés.

7.4.3 Couches réseau et transport

Nous n'utilisons pas de réseau ni de protocole de transport puisque nous nous concentrons sur les performances de la seule couche DTLS. Les protocoles réseau et transport ne sont pas requis pour un réseau où les nœuds sont à un saut du routeur (ou entre eux), donc l'adressage avec l'en-tête MAC est suffisante.

7.4.4 Implémentation de DTLS

Notre code est basé sur TinyDTLS qui est une bibliothèque implémentant la version actuelle du protocole DTLS (1.2). TinyDTLS implémente deux suites de cryptographie, la première est Counter with CBC-MAC (CCM) avec 8 octets de Message Authentication Code (MAC), couplé à AES

	Chiffrement Logiciel	Chiffrement Matériel
16 octets	484 μs	96 μs

TABLE 7.1 – Comparaison entre le chiffrement AES logiciel et matériel sur un bloc de 16 octets.

128 bits pour le chiffrement symétrique, un partage de clés basé sur un échange Diffie-Hellman éphémère et des courbes elliptiques (ECDHE) ainsi qu'un système d'authentification par DSA (également avec des courbes elliptiques) : TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8. La seconde suite cryptographique change le système d'authentification en supposant la clé pré-configurée sur les nœuds (PSK) : TLS_PSK_WITH_AES_128_CCM_8. Les implémentations de Rijndael (AES) et de SHA256 proviennent du projet OpenBSD.

7.5 Chiffrement matériel

Lors de la mesure des performances réalisée à la section suivante, nous utiliserons du chiffrement matériel que nous présentons ici. Le standard IEEE 802.15.4 nécessite un chiffrement AES pour sécuriser les communications sur le lien, il est probable qu'une implémentation matérielle d'AES soit disponible sur de nombreux appareils compatibles avec le standard. Un matériel spécialisé dédié au chiffrement permet un calcul plus rapide.

Le tableau 7.1 montre le résultat d'un test préliminaire du chiffrement matériel avec notre microcontrôleur. Le chiffrement de 16 octets avec AES 128 bits sur du matériel spécialisé est cinq fois plus rapide qu'avec du chiffrement logiciel. Le mécanisme de chiffrement CCM recommandé par le profil DICE utilise AES intensivement. Lors de nos mesures de performances, nous verrons le gain obtenu à l'établissement de connexion et sur les durées de transmission.

7.6 Mesures des performances

Dans cette section, nous présentons les résultats de notre expérimentation.

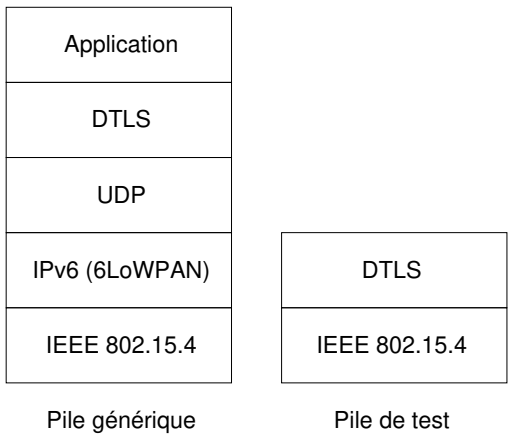


FIGURE 7.3 – Couche classique de protocoles dans les réseaux déployés, et notre pile pour mesurer uniquement l'impact de DTLS sur les communications.

Nous réduisons la pile de communication au strict minimum lors de nos mesures pour analyser au mieux l'influence de nos modifications sur DTLS, tel que montré sur la figure 7.3. Les couches réseau,

	DTLS	DTLS Optimisé	DTLS C.M.	DTLS Optimisé C.M.	Sans Sécurité
RAM (octets)	11 216	10 388	7 120	6 292	897
Flash (octets)	48 966	46 610	42 208	39 852	8 000
durée médiane poignée de mains (μs)	233 214	186 660	220 840	173 628	
durée médiane (1 octet) (μs)	18 720	18 248	11 220	10 748	4 780
durée médiane (87 octets) (μs)	43 664	43 664	21 488	21 168	10 600

TABLE 7.2 – Comparaison entre DTLS, DTLS optimisé, leur variante avec chiffrement matériel et sans DTLS ni chiffrement.

transport et application ne sont pas présentes afin de mesurer l'impact de DTLS sur un réseau IEEE 802.15.4.

Le test effectué est un aller-retour de messages, pour mesurer les durées d'échange de messages entre deux appareils (ou applications) sans synchronisation fiable du temps sur ces machines. Les tailles des messages sont variables, allant de 1 à 87 octets. Un message d'un seul octet représente le plus petit message utile (une température par exemple sans protocole comme CoAP), un message de 87 octets représente le plus grand message possible sur IEEE 802.15.4 avec un en-tête de couche liaison de 12 octets et l'en-tête DTLS de 29 octets. Nous avons essayé plusieurs configurations : DTLS standard (simple portage de DTLS sur ATmega128RFA1), une version optimisée comme décrit dans ce chapitre, puis ces deux configurations mais en activant le chiffrement matériel, et enfin un échange non sécurisé (sans DTLS ni chiffrement) pour mettre les résultats en perspective par rapport au coût de la sécurité sur un appareil contraint.

7.6.1 Mémoire Flash et RAM

Le tableau 7.2 résume les résultats observés. Premièrement, le coût en RAM pour la sécurité est plus de 10 Ko, et presque 41 Ko en Flash. La version optimisée nécessite moins de ressources (7.3% moins de RAM et 5% moins de Flash), ce qui s'explique par la suppression des messages (et le code associé). La plus grande différence est entre le chiffrement matériel et logiciel, la bibliothèque de chiffrement est optimisée pour la vitesse en déroulant les boucles ce qui implique plus d'instructions. Par ailleurs, l'algorithme AES est retiré avec le chiffrement matériel, ce qui réduit la taille du binaire de 4 Ko, ce qui est utile quand le code de l'application est large et cela doit être considéré pour des applications complexes sur des appareils très contraints.

7.6.2 Vitesse de poignée de mains

La durée de la poignée de mains DTLS est mesurée sur le client, depuis le premier message jusqu'au message Finished. La médiane des durées observées de poignée de mains faite sur 100 connexions est de 233 214 μs avec DTLS, et de 186 660 μs avec notre version optimisée. Le gain est de 46,5 ms, soit 19,9 % de la durée de l'établissement de connexion : la suppression des messages diminue le nombre d'échanges, et donc diminue la durée totale des attentes CSMA-CA.

7.6.3 Vitesse des échanges de données

La suppression de données redondantes comme expliqué en section 7.3.2 implique un temps de transmission plus rapide car le message est plus petit. De plus, cette suppression laisse plus de place aux données applicatives, ce qui contribue également à diminuer leur fragmentation.

La figure 7.4 montre nos résultats sur 3000 essais pour chaque configuration. La différence de temps d'échange aller-retour d'un message entre DTLS et notre version optimisée est visible avec un message d'un octet car la différence de taille de messages transmis est proportionnellement plus importante. Notre optimisation apporte un gain de 2,6 % de temps d'échange pour un message avec une donnée utile d'un octet. La dernière configuration ne chiffre pas la donnée, et n'utilise pas DTLS, le RTT est

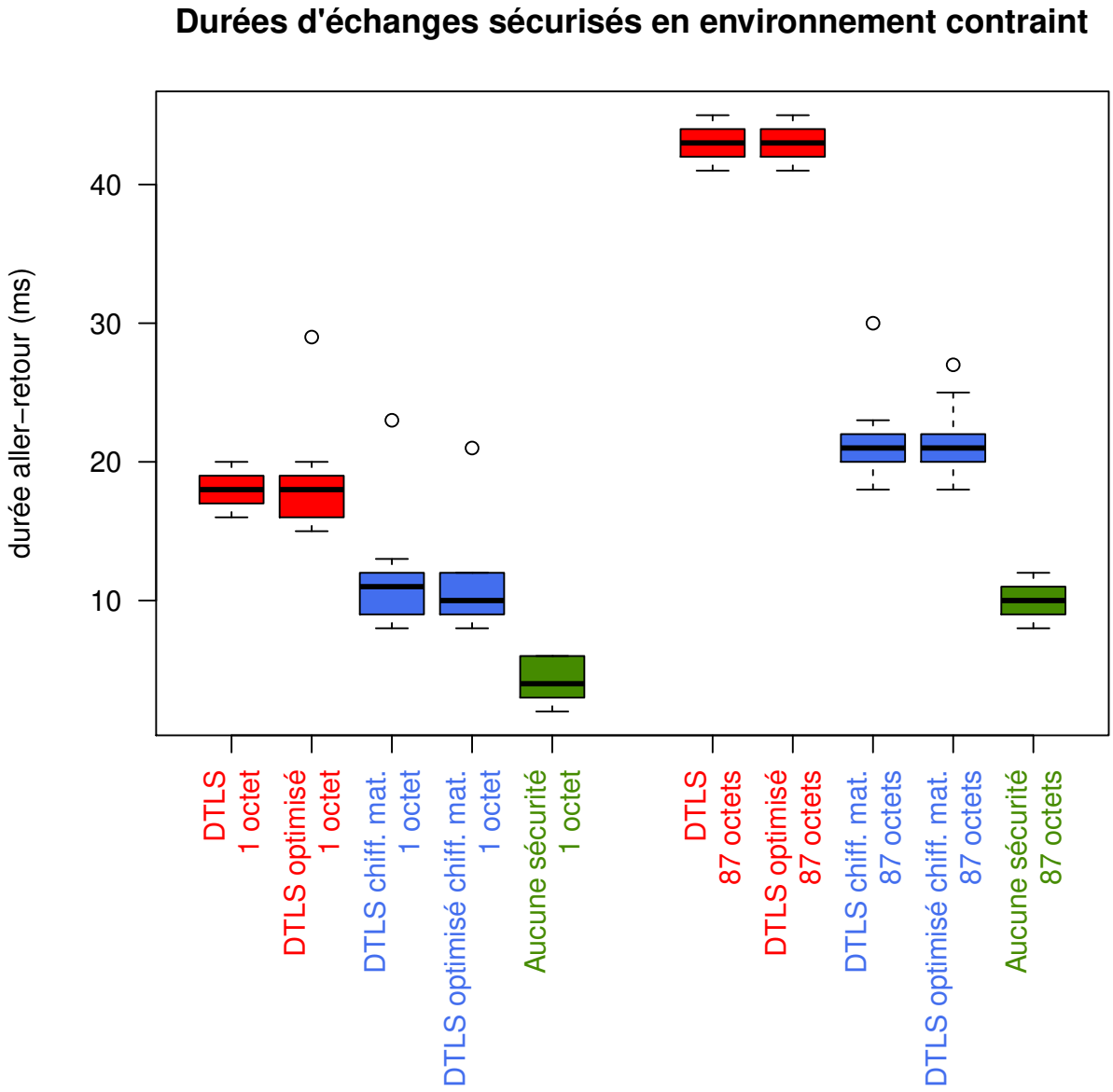


FIGURE 7.4 – Aller-retour de messages, avec un seul octet puis 87 : DTLS, DTLS optimisé, leur variante avec chiffrement matériel et enfin sans DTLS.

3,9 fois plus court, donc le temps de chiffrement et la surcharge de DTLS dans les tailles des paquets ralentit significativement la communication.

L'en-tête DTLS du message peut être réduite de 8 octets, limitant ainsi la fragmentation des messages. Par exemple, l'envoi de 1700 octets représente 18 messages avec la version optimisée de DTLS, alors qu'il est nécessaire d'envoyer 20 messages avec la version originale.

Les différentes améliorations peuvent avoir du sens selon l'application. Par exemple, l'optimisation de la poignée de mains diminue la durée de connexion : cela aide la connexion avec des nœuds mobiles (drones, voitures, etc). Il y a une légère amélioration au niveau de la latence (sur des paquets de faible taille) grâce à la suppression du vecteur d'initialisation, donc l'intérêt principal est de réduire la fragmentation.

7.7 Travaux similaires

Lithe [85] adopte une autre approche pour réduire la surcharge de DTLS : il repose sur 6LoWPAN pour compresser les informations de DTLS et ainsi gagner 8 octets de charge utile lors des échanges de données (une fois l'établissement de la connexion terminée). Le gain est rendu possible en tronquant respectivement les champs Epoch et Sequence Number à 8 et 16 bits (au lieu de 16 et 48 bits). Cette réduction de taille de champs limite le nombre d'échanges lors d'une connexion (jusqu'à 2^{16}). Par ailleurs, Lithe ne supprime pas les informations redondantes entre les couches Record et Application des en-têtes DTLS, et est lié à 6LoWPAN et est, par définition, dépendant de la couche réseau.

Robust Header Compression (ROHC) est une méthode décrite dans [86]–[90] pour compresser les en-têtes, telles que IP et UDP (entre autres). Le principe est de ne pas envoyer d'en-tête s'il n'a pas changé entre deux échanges, ou juste envoyer la différence. Cela peut être utilisé pour augmenter la charge utile d'un message, ainsi qu'avec notre version optimisée de DTLS si le déploiement inclut une couche réseau.

Les performances des courbes elliptiques sur des capteurs ont été testées [30] avec une bibliothèque nommée TinyECC. Cette bibliothèque permet aux développeurs d'activer à loisir des optimisations dans des algorithmes de courbes elliptiques, et peut être utilisée pour améliorer les performances. Ces travaux montrent que les algorithmes asymétriques avec courbes elliptiques restent un ordre de grandeur plus lents que les algorithmes symétriques sur les appareils contraints, même en étant optimisés. De plus, la taille des clés utilisées dans ces travaux est obsolète selon le NIST, les temps de calcul présentés doivent être revus à la hausse. Par conséquent, les algorithmes asymétriques ne peuvent pas êtres utilisés sur des matériels contraints sans optimisation matérielle.

Une autre approche est d'exécuter les calculs d'authentification sur un autre matériel, comme proposé par [91]. Ces travaux s'approchent d'une architecture de type maître-esclave, dans le sens où nous pouvons considérer l'entité d'authentification comme étant séparée de l'entité serveur de ressources. L'authentification peut se faire via des algorithmes asymétriques sans pénalité sur le temps d'authentification. Cependant, cette approche conserve les inconvénients caractéristiques d'une communication de bout-en-bout : empreinte mémoire liée au nombre de clients, absence de cache de données et donc nombre de messages dans le réseau contraint important.

Finalement, le nombre de messages échangés durant la poignée de mains implique des performances dégradées dans les réseaux avec rapport cyclique. Dans ces réseaux la durée de poignée de mains dépasse les 30 secondes, et notre optimisation peut grandement aider à augmenter les performances.

7.8 Conclusion

Notre travail d'optimisation de DTLS permet une amélioration de la durée des poignées de mains de 19,2 %, une réduction de la mémoire utilisée (7,3 % RAM, 5 % Flash) en supprimant des messages, et enfin un gain de 8,2 % de charge utile pour les données applicatives, grâce à la suppression du vecteur d'initialisation pour les algorithmes AEAD. Ces optimisations sur le protocole de sécurité représentent une amélioration de la communication entre un client et un serveur de ressources dans une architecture de référence. L'usage d'algorithmes asymétriques n'est cependant toujours pas envisageable, seule la séparation de l'entité d'authentification sur un matériel dédié le permettrait.

Dans le chapitre suivant, nous présenterons une architecture maître-esclave. Cette architecture sera ensuite utilisée pour mettre en avant les avantages et les inconvénients d'une architecture maître-esclave par rapport à une architecture de référence.

Chapitre 8

Exemple d'architecture maître-esclave : CASAN

Dans les chapitres précédents, nous avons étudié des architectures de sécurité avec un serveur mandataire gérant l'entité d'authentification. Cette entité d'authentification est séparée du serveur de ressources pour réduire l'empreinte mémoire et diminuer les calculs sur les nœuds contraints. Dans ce chapitre, nous présentons l'architecture maître-esclave CASAN : son fonctionnement, l'authentification des clients, ses principes de fonctionnement, la gestion de la confiance ainsi que la protection des communications. Nous introduisons également les différents inconvénients de la protection des communications avec DTLS dans CASAN. Enfin, nous présentons notre contribution : CASAN-S, un nouveau protocole de communication intégré à l'architecture CASAN, plus efficace et léger, qui simplifie la communication entre le maître et ses esclaves. Cette optimisation est permise grâce à l'architecture maître-esclave qui ne nécessite pas une communication de bout-en-bout, et permet par conséquent d'alléger les échanges dans le réseau contraint.

Ce chapitre présente l'architecture maître-esclave CASAN, son fonctionnement, l'authentification des clients, ses avantages et la protection de ses communications. De plus, nous décrivons cette architecture selon notre méthode de description des architectures de sécurité présentée au chapitre 4 (page 29).

Ce chapitre présente les contributions suivantes :

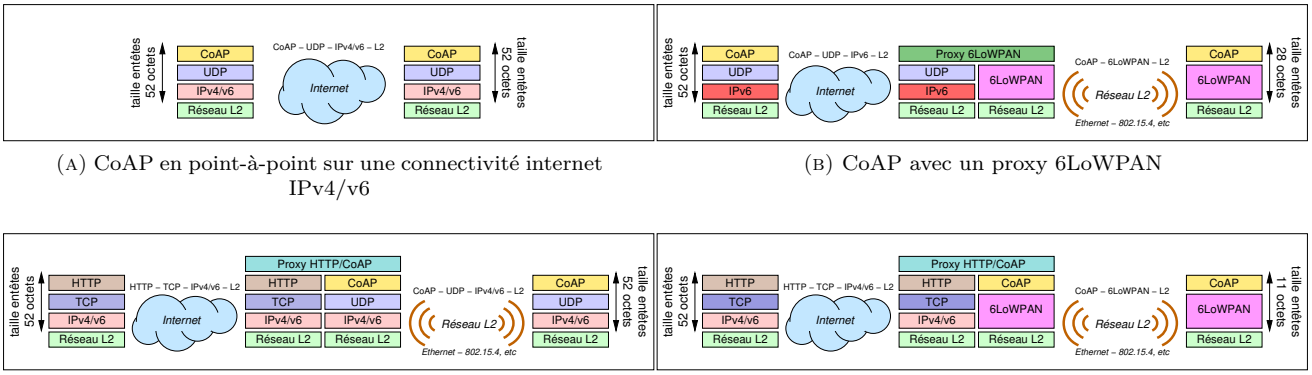
1. L'intégration de la sécurité dans les échanges CASAN, menant à la création du protocole CASAN-S ;
2. La description de cette nouvelle architecture selon la méthode décrite au chapitre 4 (page 29).

8.1 Architectures de référence

La vision de l'Internet des Objets promue par l'IETF met en avant les protocoles 6LoWPAN et CoAP.

Les schémas 8.1 illustrent cette vision, avec des cas d'usage du protocole CoAP selon une logique d'architecture de référence : dans chaque cas, la connexion est de bout-en-bout, du client au serveur de ressources. La prise en compte des contraintes dans les réseaux de capteurs est croissante. La taille minimale des messages associée aux cas d'usage est indiquée sur le schéma [1]. Dans chaque scénario, le capteur (sur la droite) embarque une pile IP complète (IPv6 ou IPv4 et IPv6). La taille des paquets sur le réseau est réduite avec l'aide de 6LoWPAN. Il faut une pile IP car l'adressage est commun (les deux utilisent une adresse IP pour se contacter). Ces illustrations ne présentent pas les protocoles associés (DNS, détection d'adresses dupliquées, découverte des voisins, etc.), ni les mécanismes nécessaires en

1. La taille de l'en-tête de niveau 2 n'est pas indiquée puisqu'elle dépend de la technologie utilisée. Dans chaque cas, nous comptons la taille la plus favorable, par exemple nous comptons 32 octets = 20 (IPv4, pas IPv6), 8 (UDP) et 4 (en-tête CoAP sans option ni token).



(A) CoAP en point-à-point sur une connectivité internet IPv4/v6

(B) CoAP avec un proxy 6LoWPAN

(C) capteur IPv4/v6 avec un proxy HTTP

(D) capteur 6LoWPAN avec proxy HTTP

FIGURE 8.1 – Cas d'usage de CoAP. Le client est à gauche, le capteur est à droite. Les protocoles ainsi que la taille des messages sont indiqués.

pratique dans les déploiements (comme le contrôle d'accès). Ces usages n'ont pas engendré le succès escompté puisqu'il n'y a à ce jour ni offre commerciale ni déploiement significatif.

Nous avons montré des architectures de référence, ainsi qu'une architecture maître-esclave (MQTT-SN) mais qui ne permettait pas un déploiement d'actionneurs. Nous souhaitons désormais illustrer une architecture maître-esclave qui comble ce besoin : CASAN.

8.2 Introduction à l'architecture maître-esclave CASAN

Dans cette section, nous présentons CASAN : ses objectifs, son fonctionnement et les implications d'une architecture maître-esclave sur les contraintes des matériels et du réseau.

CASAN (pour "Common Architecture for Sensor and Actuator Networks") [92] est une architecture conçue pour les environnements contraints. Ses objectifs sont, premièrement, de minimiser la complexité des nœuds dans les environnements contraints. Deuxièmement, l'architecture doit gérer la mise à l'échelle pour accepter des réseaux très denses et avec potentiellement de nombreux clients. Troisièmement, l'architecture ne doit pas imposer de technologie de niveau 2, et doit fonctionner par exemple sur des réseaux aussi différents que IEEE 802.15.4 et Ethernet.

Pour atteindre ces objectifs, l'architecture CASAN place les entités d'authentification et d'autorisation sur un appareil spécialisé, diminuant la charge de calcul et de mémoire sur les nœuds contraints. Cette architecture fournit également une interface REST aux clients sur Internet, permettant un chaînage des appareils.

Par effet de bord, la séparation des entités permet l'usage d'un matériel non contraint sur le réseau, par lequel passeront toutes les requêtes. Sur ce matériel, nous souhaitons ajouter une fonctionnalité de traduction de requêtes vers le protocole utilisé au sein du réseau contraint, par exemple de HTTP vers CoAP. L'architecture est alors accessible au plus grand nombre de clients possible, permettant à des clients implémentant des piles de protocoles variées d'accéder de manière transparente aux serveurs de ressources.

La figure 8.2 montre un aperçu de l'architecture CASAN. Les clients interagissent en HTTP (ou CoAP) sur Internet avec un équipement non contraint nommé « maître », qui :

1. concentre les entités d'authentification et d'autorisation ;
2. reçoit les requêtes, qui lui sont adressées directement, sous la forme d'URI telles que `/temp1/value` ;
3. fait une traduction des requêtes aux esclaves.

La pile de communication utilisée dans le réseau contraint est indépendante du réseau IP, et n'utilise que CoAP directement sur une couche liaison. Par conséquent, les esclaves n'ont pas à gérer

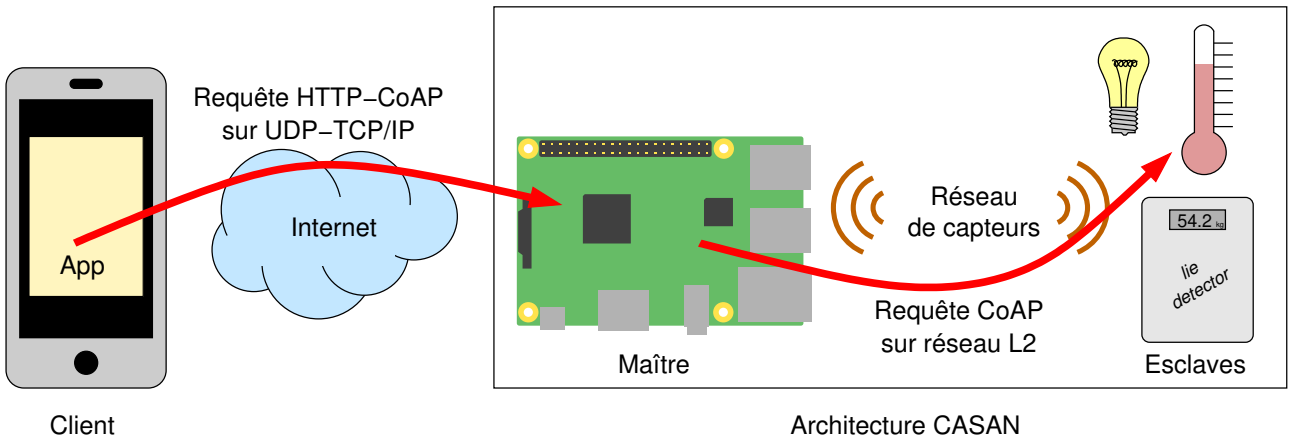


FIGURE 8.2 – Aperçu de CASAN

d'états liés aux couches réseau et transport. Le code est alors simplifié, peu de mémoire est nécessaire pour retenir des états, et la taille des paquets échangés est réduite.

CASAN est donc une architecture maître-esclave : les requêtes sont à destination du maître, qui s'occupe des clients (authentification, traduction des requêtes), et les esclaves n'ont pas conscience des clients.

Les fonctionnalités impliquées par une architecture maître-esclave sont :

1. la présence d'un cache de données (si les données s'y prêtent) ;
2. la découverte de ressources à faible coût : le maître garde une base de données des ressources de chaque esclave, qui sont ensuite fournies par le système CASAN aux clients ;
3. le client et les esclaves n'ont pas à se connaître : le client peut demander la liste des ressources au maître, puis demander les ressources selon la sémantique qui est associée à leur nom, sans associer le nom de la ressource à un serveur de ressources. Par exemple, la ressource peut être nommée « températureSalleCafé », son nom est indépendant de sa localisation, par conséquent nous pouvons changer la ressource de réseau de manière transparente, sans interruption de service. De leur côté, les esclaves ne connaissent que le maître.

Les esclaves ne communiquent pas directement avec les clients, la pile de communication peut donc être simplifiée. Les protocoles génériques (IP et UDP) ont des fonctionnalités non nécessaires dans notre cas, qui sont retirés de la pile de communication des esclaves : la couche réseau n'est pas nécessaire, la communication n'est pas de bout en bout et se limite à deux appareils dans le même réseau. La couche transport n'est pas non plus nécessaire, l'esclave répond uniquement aux requêtes du maître, l'usage de ports réseau n'est donc pas nécessaire. À l'inverse, la couche applicative reste nécessaire pour les requêtes, nous l'utilisons pour exprimer une demande (type de requête, sujet de la requête, nécessité d'avoir une réponse, etc.). Toutes les données du système CASAN sont transmises via le protocole CoAP directement sur la couche liaison ; la taille minimale d'un message CASAN est donc de seulement 4 octets sur une couche liaison.

Nous pouvons remarquer que la communication n'étant pas entre le client et le serveur de ressources directement, la protection des communications ne sera elle-même pas de bout-en-bout (voir section 3.1.7). Cependant, nous faisons confiance au maître : il est dans notre domaine de sécurité, ce n'est pas un serveur distant contrôlé par un tiers. Le maître gère par la suite la confiance avec le reste des nœuds du réseau, et en cas de changement de confiance (changement de clés de chiffrement) sur un nœud, il est le seul impacté : la fin de vie d'un nœud n'implique pas de changer les clés sur tous les clients qui lui faisaient confiance. Cette remarque vaut pour toute architecture permettant la mise en cache d'une donnée en clair.

En conclusion, CASAN est une architecture maître-esclave : les clients et les serveurs de ressources ne communiquent pas directement ensemble, et l'authentification et le partage de clés s'effectuent sur

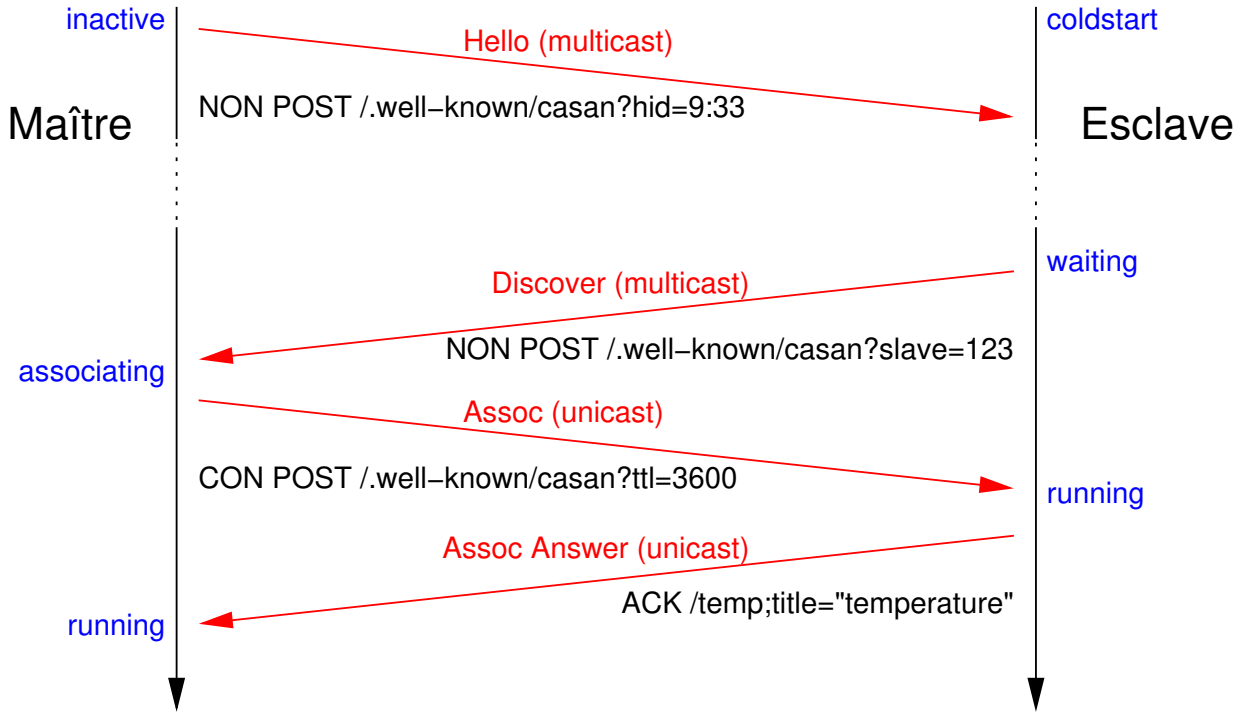


FIGURE 8.3 – Mécanisme d'association CASAN. En bleu les états du protocole CASAN, en rouge les messages CASAN, et en noir les éléments des messages CASAN.

un nœud maître adapté aux opérations cryptographiques, y compris asymétriques. De plus, l'architecture fournit un cache de données, une liste de ressources et la découverte de ressources s'effectue une seule fois au démarrage des nœuds afin de réduire au maximum l'empreinte réseau. La communication n'étant pas de bout-en-bout, la pile de protocoles ainsi que le nombre de messages sont réduits sur le réseau contraint. De plus, le client fait confiance au maître qui gère la confiance avec le reste des nœuds du réseau.

8.3 Principes de CASAN

Dans cette section, nous présentons le mécanisme d'association de CASAN, les messages échangés dans l'architecture ainsi que le code associé à l'implémentation.

Puisqu'un maître CASAN doit transmettre une requête au bon esclave, il doit savoir quelles sont les ressources associées à chaque esclave dans son domaine. Pour faire cela, l'architecture CASAN dispose de deux mécanismes.

Le premier est l'*amorçage* : il doit être effectué une seule fois dans la vie d'un esclave, pour échanger les identités et d'autres éléments comme le matériel cryptographique. Le second mécanisme est l'*association* : illustrée à la figure 8.3, l'association est effectuée à chaque démarrage du client ou du maître. Le maître envoie périodiquement des messages pour s'annoncer avec un identifiant unique "hello id". Quand un esclave démarre, il envoie un message Discover avec son identité et le maître répond avec une requête d'association contenant une durée d'expiration (Time To Live). L'esclave donne ensuite la liste de ses ressources au format `/.well-known/casan` qui est similaire au format CoAP[93]. Après cet échange, l'association reste valide pour la durée indiquée par le maître, et doit être renouvelée si besoin avant expiration. Quand le maître redémarre, il change son identifiant "hello id". L'esclave doit détecter ce message et démarrer une nouvelle association.

Utiliser uniquement des mécanismes simples et retirer les protocoles qui ne sont pas nécessaires diminue la complexité du logiciel comparé aux implémentations traditionnelles de CoAP sur IP [92].

L'implémentation du maître CASAN a été testée sur un ordinateur de type Raspberry Pi avec un système d'exploitation générique et un maître écrit en 2800 lignes de code Python. L'implémentation de l'esclave CASAN a été testée sur une plateforme de type Arduino (micro-contrôleur ATmega avec seulement 2 Ko de SRAM) pour l'esclave, et le code C++ ne fait que 3000 lignes.

8.4 Protection des communications dans CASAN

Dans cette section, nous décrivons CASAN selon la définition formelle et la méthode de description fournies au chapitre 4 (page 29), puis nous présentons les avantages et inconvénients de l'architecture CASAN.

Les communications sont protégées par DTLS entre le maître et ses esclaves, et par DTLS ou TLS entre le client et le maître, selon que le client utilise CoAP ou HTTP. Dans nos exemples, nous considérons un client qui se connecte en HTTP.

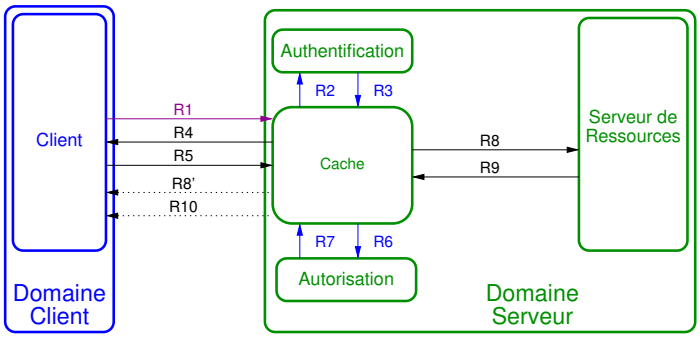


FIGURE 8.4 – Modèle de l'architecture CASAN.

La figure 8.4 représente le modèle de l'architecture CASAN selon notre formalisation définie dans le chapitre 4. Deux domaines sont définis :

— $Domaine(client) = \{C\}$
— $Domaine(serveur) = \{Cache, AN, AZ, SR\}$

Le tableau 8.1 et la figure 8.4 décrivent les relations dans l'architecture CASAN lors d'une requête. Les relations sont :

1. (R_1 et R_4) connexion TLS (abrégée pour lisibilité) ;
2. (R_5 à R_7) requête puis autorisation de la requête ;
3. Ensuite, deux cas se présentent :
   (a) (R_8 à R_{10}) requête CASAN au serveur de ressources puis réponse au client ;
   (b) (R'_8) la donnée est déjà en cache sur le maître, la réponse est alors directe au client.

CASAN est noté comme un protocole à part entière dans le tableau. Les entités d'authentification, d'autorisation et de cache sont liées, et constituent en partie le maître.

	entités	protocoles	propriétés	informations
R_1	C ↔ Cache	TLS	F, O, C, I, MA, A, R, AC	id_C, preuve id_C, id_{AN}, preuve id_{AN} *connexion TLS abrégée (lisibilité)*
R_2	Cache → AN	⊥	⊥	⊥
R_3	AN → Cache	⊥	⊥	⊥
R_4	Cache → C	TLS	F, O, C, I, MA, A, R, AC	fin établissement connexion TLS
R_5	C → Cache	TLS, HTTP	C, I, MA, R	requête
R_6	Cache → AZ	⊥	⊥	⊥
R_7	AZ → Cache	⊥	⊥	autorisation
R_8	Cache → SR	DTLS, CASAN	C, I, MA, R	requête
R'_8	Cache → C	TLS, HTTP	C, I, MA, R	réponse (si la donnée est en cache)
R_9	SR → Cache	DTLS, CASAN	C, I, MA, R	réponse
R_{10}	Cache → C	TLS, HTTP	C, I, MA, R	réponse

TABLE 8.1 – Relations de l'architecture CASAN, avec une requête HTTPS du client.

L'architecture CASAN présente plusieurs avantages par rapport à une architecture de référence. Premièrement, le serveur de ressources est allégé. Il est allégé en calculs, puisqu'il n'a qu'une seule connexion à établir, celle avec son maître. Il est également allégé en mémoire, puisqu'il n'a pas à connaître les clients. Le serveur de ressources n'a donc pas à retenir d'informations comme des identifiants ou des secrets avec les clients. Il n'a pas non plus de communication avec eux, donc pas de contexte de connexion (clés de chiffrement, algorithmes, nombre de messages échangés, etc.).

Deuxièmement, le réseau contraint a sa propre pile de communication, distincte de celle utilisée pour communiquer avec le client, et cette pile de protocoles n'utilise que CoAP et DTLS, directement sur une couche liaison. Par conséquent, le code est peu complexe et ne nécessite que peu de mémoire pour la pile de communication, et la taille des paquets échangés est réduite par rapport à une pile de communication incluant les couches réseau et transport.

Troisièmement, CASAN permet un cache de données, ce qui n'est pas possible dans une architecture de référence.

Quatrièmement, CASAN permet la découverte de ressources à faible coût, grâce au maître qui connaît la liste des ressources des esclaves. Les esclaves ne sont donc pas sollicités en dehors de leur connexion au maître. Dans une architecture de référence, il faut ajouter un service supplémentaire qui récoltera cette liste, le « répertoire de ressources » [94].

Cinquièmement, le maître est le seul nœud auquel le client doit faire confiance, le client n'est alors pas concerné en cas de compromission ou de fin de vie d'un capteur et ne nécessite pas une mise à jour de ses clés. Le maître est responsable du maintien de la confiance avec les serveurs de ressources (c'est-à-dire maintenir des secrets partagés avec eux) et de la vérification de leur intégrité.

Enfin, le client ne connaît que le maître et n'interagit qu'avec lui. Le maître n'est pas contraint et il peut traduire des requêtes pour les esclaves. Par conséquent, les clients choisissent les protocoles qu'ils souhaitent pour récupérer les données (exemple : HTTPS ou CoAPS).

CASAN a deux inconvénients. Premièrement, le serveur de ressources maintient une connexion avec le maître. Cette connexion nécessite de réveiller de temps en temps les esclaves pour la maintenir. Ce fonctionnement n'est cependant pas intrinsèque à toutes les architectures maître-esclave, qui pourraient utiliser des objets de sécurité à la place d'une communication DTLS, par exemple. Deuxièmement, dans CASAN, la communication n'est pas sécurisée de bout-en-bout. Les communications sont toutes sécurisées mais le maître connaît la ressource échangée. Cacher l'information au maître reviendrait à partager un secret entre le client et les serveurs de ressources, ce qui va à l'encontre du principe même d'une architecture maître-esclave et annule ses avantages.

8.5 CASAN et DTLS

Dans cette section, nous présentons la gestion de la confiance ainsi que la protection des communications dans l'architecture CASAN.

L'architecture CASAN couvre deux aspects de la sécurité : la confiance et la protection des communications.

8.5.1 Confiance

Pour considérer un objet comme un esclave CASAN, le réseau doit lui faire confiance, c'est-à-dire partager un secret avec cet objet.

Avant la première connexion d'un esclave à son maître, les deux doivent déjà partager un secret. Ce secret est partagé via un canal sécurisé, hors bande, c'est-à-dire par un moyen autre qu'une communication sur le réseau physique habituel. Par exemple, l'architecture CASAN s'attend à ce que ces informations soient partagées au préalable avec une technologie comme Near-Field Communication (NFC).

Ensuite, ce secret est utilisé pour sécuriser les communications : il authentifie les pairs et est utilisé pour générer des clés effectives de chiffrement et d'authentification, uniques à chaque session. La création de clés effectives de chiffrement à partir d'une clé pré-chargée de long terme est identique au fonctionnement de TLS, DTLS, IPsec, etc.

La communication entre le maître et ses esclaves est donc validée par un opérateur humain, via l'échange hors bande d'un secret, et les communications sont protégées également par ce secret.

8.5.2 Communication

Entre le client et le maître, la pile de protocoles utilisée est celle des communications sur Internet en général. Les deux matériels sont non contraints, il n'est pas nécessaire de recourir à une mesure particulière pour économiser des ressources.

Les esclaves sont contraints, et la pile de communication est uniquement basée sur le protocole CASAN (lui-même basé sur CoAP) et une couche de sécurité. Le protocole DTLS est utilisé dans l'architecture CASAN avant que le maître n'envoie son message d'association, puis les messages suivants sont protégés : la liste des ressources reste confidentielle.

8.6 DTLS n'est pas suffisant

Dans cette section, nous introduisons les problèmes associés à la protection des communications avec DTLS dans CASAN, car DTLS souffre de plusieurs inconvénients comme le nombre important de messages lors de la connexion et la taille de ses en-têtes malgré les recommandations pour les réseaux contraints par le groupe de travail IETF DICE [36].

8.6.1 Médium partagé

Dans les réseaux filaires, où les nœuds sont reliés à des commutateurs, tous les nœuds peuvent émettre des données en même temps. Le lien qui les relie au commutateur n'est pas partagé, chaque nœud a le sien, et c'est au commutateur d'ordonnancer les paquets que les nœuds envoient par la suite. Dans les réseaux sans fils, les nœuds ne peuvent pas communiquer tous à la fois, car ils échangent sur le même médium. Par conséquent, la durée et la fréquence des échanges influent sur la communication des autres nœuds, qui doivent attendre pour échanger à leur tour.

DTLS n'a pas été conçu pour les environnements avec un médium partagé. Dix messages sont échangés, et la poignée de mains dure trois RTT, soit une durée importante d'établissement de la connexion. Même sans pertes, l'établissement de la connexion prend du temps. Si le réseau est configuré avec un rapport cyclique (*duty-cycled network*), la connexion prend jusqu'à 50 secondes [83].

8.6.2 Maximum Transfer Unit limité

DTLS ajoute 29 octets à chaque message [95], ce qui représente une charge utile fortement diminuée dans les réseaux contraints. Sur un réseau IEEE 802.15.4, ces 29 octets d'en-tête représentent 23 % du MTU.

8.6.3 Pas d'intégration

La poignée de mains ne comporte que des informations de DTLS, avant l'échange de tout autre message de données. Le protocole DTLS ne prévoit pas l'échange d'autres informations publiques durant cette poignée de mains. Par conséquent, l'échange de données liées aux protocoles de couches supérieures nécessite des aller-retours supplémentaires, retardant l'échange de données applicatives. DTLS n'est donc pas adapté pour l'intégration avec d'autres protocoles.

Puisque DTLS n'est pas adapté aux communications entre le maître et ses esclaves, nous proposons un nouveau protocole de communication que nous nommons CASAN-S.

8.7 Contraintes et prérequis pour le protocole CASAN-S

Dans cette section, nous présentons les propriétés nécessaires de notre nouveau protocole de communication entre le maître et l'esclave. Ce nouveau protocole de communications sécurisées doit prendre en compte les problèmes observés dans DTLS.

Le chapitre 4 résume les propriétés de sécurité nécessaires dans une architecture de sécurité. La protection de la communication doit inclure au moins la confidentialité, l'authenticité et l'intégrité des messages, et l'architecture doit prévenir l'usurpation d'identité d'un pair. En plus de ces fonctionnalités de base, le protocole CASAN-S doit répondre aux critères suivants.

8.7.1 Connexion

Les esclaves peuvent tomber en panne ou ne plus être accessibles. Dans ce cas, pour éviter l'envoi de requêtes (avec retransmissions) sur le réseau contraint, et pour que les clients aient rapidement une réponse, nous décidons que les esclaves doivent maintenir une connexion avec leur maître. Le maître et l'esclave doivent alors échanger régulièrement pour rester informé de leur disponibilité. Par conséquent, le protocole CASAN-S est adapté pour des connexions de longue durée, évitant un renouvellement systématique des associations de sécurité à chaque nouvel échange. De plus, une poignée de mains à l'établissement de la connexion est nécessaire pour négocier des algorithmes à utiliser et le matériel cryptographique. Cette poignée de mains permet aux échanges suivants de bénéficier d'une plus grande taille de données utiles puisque le contexte est stocké sur les nœuds.

8.7.2 Fiabilité

Durant l'association, le protocole CASAN-S doit être capable d'échanger des messages de manière fiable sur un lien non fiable. Un mécanisme de retransmission est nécessaire pour s'assurer que les messages de la poignée de mains soient correctement délivrés.

8.7.3 Intégration

Le protocole de sécurité doit faire partie du protocole CASAN-S. Cette intégration permet d'échanger les données publiques pendant la négociation des paramètres de sécurité dans les mêmes messages. Au delà de la réduction du nombre de messages, cette intégration s'accorde avec la conception de CASAN dont l'objectif est de réduire l'empreinte de la pile de protocoles sur les appareils contraints.

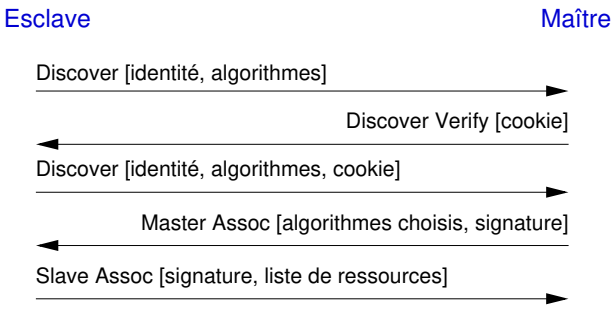


FIGURE 8.5 – Poignée de mains CASAN-S.

8.7.4 Médium partagé

Le protocole de sécurité doit être efficace en nombre de messages, pour éviter un usage excessif du médium partagé. La sobriété des échanges permet une implémentation plus simple, ce qui est bénéfique pour les nœuds contraints.

8.7.5 Contraintes

Les esclaves ont une mémoire et des capacités de calcul limitées, le protocole CASAN-S doit de ce fait être efficace dans ses opérations cryptographiques. Le maître et l'esclave partagent des clés cryptographiques dans l'architecture CASAN-S (échangées lors de la phase d'amorçage), donc la cryptographie asymétrique n'est pas nécessaire sur l'esclave [30], ce qui permet une économie de temps de calcul et de mémoire sur les esclaves (autour de 1500 octets de RAM et 10 à 15 Ko de mémoire Flash pour les optimisations [82]).

8.7.6 Services de sécurité additionnels

Au delà de la confidentialité, de l'authenticité et de l'intégrité des messages, le protocole CASAN-S doit être protégé des attaques par rejeu. Cela est particulièrement important avec des actionneurs, un message rejoué qui ouvre une porte peut mettre en danger toute l'entreprise. Les communications doivent également être protégées d'une attaque (simple) par déni de service, les nœuds malicieux doivent avoir un effet minimal sur l'architecture. Enfin, les appareils doivent pouvoir être mis à jour, pour prendre en compte les failles découvertes dans les algorithmes ou les implémentations. Le protocole de communication doit permettre aux appareils de s'accorder sur les algorithmes à utiliser (agilité cryptographique), pour permettre des mises à jour sur des réseaux hétérogènes (versions différentes de logiciel).

8.8 Conception du protocole CASAN-S

Dans cette section, nous présentons notre contribution, CASAN-S, le remplaçant du protocole CASAN qui intègre la sécurité aux échanges sans nécessiter de mécanisme comme DTLS.

La figure 8.5 représente la poignée de mains CASAN-S, remplaçant les messages Discover, Assoc et Answer de la figure 8.3, page 84. Les informations importantes sont entre crochets. Les principales différences entre CASAN-S et CASAN sécurisé par DTLS sont expliquées par la suite.

8.8.1 Intégration et cohérence

Les messages sont au format CoAP, pour être cohérent avec l'architecture CASAN. La poignée de mains négocie des algorithmes et du matériel cryptographique, et échange des données relatives

à l'architecture CASAN-S, telles que les identités et la liste des ressources des esclaves. Les deux premiers messages concernent la protection contre le déni de service (tel que dans DTLS). Ensuite, le message Master Assoc contient les algorithmes choisis, la signature de la poignée de mains pour s'assurer que la connexion n'a pas été altérée et des informations chiffrées pour l'esclave (telle que la durée d'expiration de la connexion). Enfin, le message Slave Assoc contient la signature de la poignée de mains et la liste chiffrée des ressources de l'esclave.

8.8.2 Fragmentation

La fragmentation consiste à séparer les données en différents paquets quand le contenu est trop large, c'est une fonctionnalité importante dans les réseaux contraints où le MTU est faible. Le protocole CoAP a déjà un mécanisme pour fragmenter les messages avec des options « Block » [96], il n'est donc pas nécessaire de répliquer cette fonctionnalité dans le protocole CASAN-S.

8.8.3 Numéros de Séquence

Pour chiffrer plusieurs messages avec une seule clé, il est nécessaire d'utiliser un numéro de séquence. La taille du numéro de séquence choisi est de 16 bits, soit 65536 messages, ce qui est suffisant dans un réseau contraint où le nombre de messages échangés est faible entre deux redémarrages. Si le numéro de séquence dépasse sa valeur maximale, la connexion peut être renouvelée.

8.8.4 Sessions

La communication maître-esclave peut être redémarrée, ce qui implique de pouvoir informer l'interlocuteur d'un redémarrage. Nous implémentons un numéro de session sur 8 bits. Les sessions sont indépendantes, et chaque nouvelle session nécessite un nouveau numéro de session pour distinguer une nouvelle connexion. Chaque nouvelle connexion remplace automatiquement la session précédente, une seule connexion est active à la fois entre un maître et un esclave. Par conséquent, il n'est pas nécessaire d'avoir un grand nombre de numéros de sessions.

Pour résumer, l'architecture CASAN-S ajoute un nouveau protocole de communication cohérent (utilisation de CoAP uniquement, déjà présent dans CASAN) et efficace (peu de messages échangés) entre le maître et l'esclave. Le protocole CASAN-S représente une seule couche avec sécurité intégrée, basée uniquement sur le format CoAP, indépendant de la couche liaison, adapté aux contraintes de taille des données dans les environnements contraints et ne nécessitant que 5 messages pour une association au lieu des 13 avec le protocole CASAN protégé par DTLS.

8.8.5 Conclusion

Les échanges CASAN-S sont au format CoAP et contiennent à la fois la négociation des algorithmes cryptographiques à utiliser et des paramètres de communication (comme le TTL) pour ne pas nécessiter d'aller-retours supplémentaires. La liste des ressources disponibles sur l'esclave est envoyée chiffrée au maître, comme dans CASAN, mais cela se fait en seulement 5 messages et 2 RTT au lieu de 13 messages et 5 RTT (10 échanges DTLS puis 3 échanges CASAN). La fragmentation est gérée par CoAP. La taille des numéros de séquence est un compromis entre le souhait de réduire la taille des en-têtes et la fréquence de renouvellement de la connexion.

8.9 CASAN-S formalisé

Dans cette section, nous décrivons CASAN-S selon la définition formelle et la méthode de description fournies au chapitre 4 (page 29). Le formalisme de CASAN-S est proche de celui de CASAN, et n'implique qu'un changement mineur dans les relations.

La figure 8.6 représente le modèle de l'architecture CASAN-S. Deux domaines sont définis :

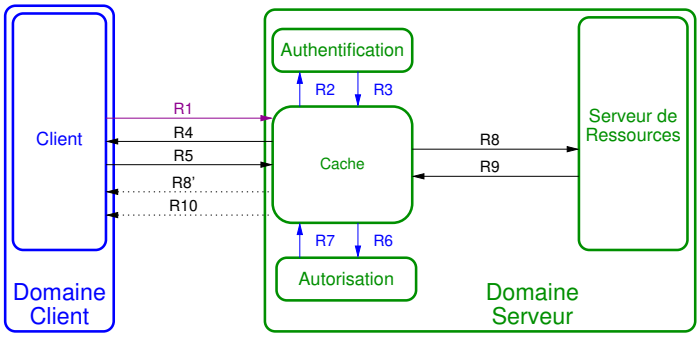


FIGURE 8.6 – Modèle de l'architecture CASAN-S, identique au modèle de l'architecture CASAN.

— $Domaine(client) = \{C\}$
— $Domaine(serveur) = \{Cache, AN, AZ, SR\}$

	entités	protocoles	propriétés	informations
R_1	C ↔ Cache	DTLS	F, O, C, I, MA, A, R, AC	id_C, preuve id_C, id_{AN}, preuve id_{AN} *connexion DTLS abrégée (lisibilité)*
R_2	Cache → AN	⊥	⊥	⊥
R_3	AN → Cache	⊥	⊥	⊥
R_4	Cache → C	DTLS	F, O, C, I, MA, A, R, AC	fin établissement connexion DTLS
R_5	C → Cache	DTLS, CoAP	C, I, MA, R	requête
R_6	Cache → AZ	⊥	⊥	⊥
R_7	AZ → Cache	⊥	⊥	autorisation
R_8	Cache → SR	CASAN-S	C, I, MA, R	requête
R'_8	Cache → C	DTLS, CoAP	C, I, MA, R	réponse (si la donnée est en cache)
R_9	SR → Cache	CASAN-S	C, I, MA, R	réponse
R_{10}	Cache → C	DTLS, CoAP	C, I, MA, R	réponse

TABLE 8.2 – Relations de l'architecture CASAN-S, avec une requête CoAPS du client.

Le tableau 8.2 et la figure 8.6 décrivent les relations dans l'architecture CASAN-S lors d'une requête. Les relations sont :

1. (R_1 à R_4) connexion DTLS (abrégée pour lisibilité) ;
2. (R_5 à R_7) requête puis autorisation de la requête ;
3. Ensuite, deux cas se présentent :
   (a) (R_8 à R_{10}) requête CASAN-S au serveur de ressources puis réponse au client ;
   (b) (R'_8) la donnée est déjà en cache sur le maître, la réponse est alors directe au client.

Les entités d'authentification, d'autorisation et de cache sont toujours liées entre elles. CASAN-S est le seul protocole à être utilisé au sein du réseau contraint (entre SR et Cache).

CASAN-S apporte deux améliorations par rapport à CASAN. Premièrement, le démarrage du réseau implique moins de messages. Deuxièmement, la charge utile maximale des messages est plus

importante. Mis à part ces changements, les avantages et inconvénients de l'architecture CASAN-S sont identiques à ceux de CASAN.

8.10 Conclusion

Une architecture maître-esclave permet la stricte séparation du réseau contraint des autres réseaux. Le réseau contraint est indépendant des échanges effectués avec le client, nous pouvons donc adapter la pile de protocoles en supprimant les couches qui ne sont nécessaires que pour une communication de bout-en-bout. Ainsi, nous disposons d'une implémentation adaptée à des nœuds fortement contraints en capacité de calcul et mémoire.

Notre contribution a été de faire évoluer l'architecture CASAN en améliorant la communication entre le maître et l'esclave, cette nouvelle architecture s'appelle CASAN-S. La communication entre le maître et l'esclave s'effectuait au départ avec DTLS, mais la conception maître-esclave de CASAN permet de réduire la complexité des échanges : toutes les fonctionnalités de DTLS n'étaient pas nécessaires, parfois redondantes avec celles de CoAP, nous les avons donc retirées pour alléger la charge sur le réseau contraint. CASAN-S permet de passer de 13 à 5 messages échangés au démarrage d'un nœud (et passer de 5 à 2 RTT) ainsi qu'un gain de charge utile dans les messages par rapport à CASAN protégé par DTLS. L'implémentation est donc plus simple, ce qui est bénéfique pour les nœuds contraints.

Dans le chapitre suivant, nous comparerons qualitativement puis nous effectuerons des mesures de performances entre une architecture maître-esclave et une architecture de référence. Nous allons vérifier, entre autres, la différence entre CASAN-S et A-DTLS en temps de démarrage de l'architecture et en temps de connexion lors d'une requête, soient les opérations courantes lors de la phase opérationnelle.

Chapitre 9

Influence d'une architecture maître-esclave

Ce chapitre illustre l'intérêt d'une architecture de type maître-esclave dans les réseaux contraints par rapport à une architecture de référence. Pour cela, notre première contribution est la comparaison des architectures CASAN, CASAN-S et A-DTLS (avec et sans cache de données) afin d'analyser qualitativement les avantages d'une architecture de type maître-esclave. Pour notre seconde contribution, nous mesurons le gain de performances obtenu lors de l'ajout d'un nœud maître non contraint (c'est-à-dire lorsque le point de terminaison sécurisé est séparé du serveur de ressources). Nous montrons qu'une architecture maître-esclave exploite mieux les ressources limitées (concernant le débit, le partage du médium, ou encore la taille des messages) qu'une architecture de référence. Pour cela nous proposons deux scénarii mettant en avant les performances dans des situations classiques de la phase opérationnelle d'une architecture : démarrage d'un réseau et requête des capteurs.

Ce chapitre présente les contributions suivantes :

1. La comparaison qualitative des architectures A-DTLS avec et sans cache de données, CASAN et CASAN-S ;
2. La mesure de la durée de démarrage du réseau et de requête d'un nœud pour les architectures A-DTLS et CASAN-S.

9.1 Comparaison qualitative

Dans cette première section, nous comparons les architectures A-DTLS, A-DTLS avec un cache de données, CASAN et CASAN-S. Le contexte et les critères de cette comparaison sont ceux du chapitre 5 (page 41).

Le tableau 9.1 montre les différences entre les architectures A-DTLS (avec et sans cache de données), CASAN, et CASAN-S. Ce tableau reprend les critères de comparaison du chapitre 5, tout en ajoutant d'autres critères plus spécifiques :

1. cryptographie asymétrique raisonnable sur le client : ce critère indique que le client peut utiliser des algorithmes asymétriques sans que cela n'ait d'impact trop important sur le serveur de ressources. Dans l'architecture A-DTLS sans cache de données, si le client utilise des algorithmes asymétriques, alors le serveur de ressources le devra également. Ce n'est pas le cas pour les autres architectures.
2. surcharge de la taille de messages sur le réseau contraint, dans le meilleur et le pire cas ;
3. taille de la charge utile : ce critère indique la taille maximale de la donnée transportée dans l'architecture en un seul message. Cette taille prend en compte la taille d'en-tête de la couche liaison.

1. Premier cas : longues adresses (IP, lien), UDP sans compression de port. Second cas : adresses IP implicites (en-tête IPv6 de 3 octets), UDP compressé (en-tête de 4 octets).

	A-DTLS	A-DTLS (avec cache)	CASAN + DTLS	CASAN-S
entités du domaine client	client	client	client	client
entités du domaine serveur	SR, AN, AZ	SR, AN, AZ	SR, AN, AZ, cache	SR, AN, AZ, cache
Sécurité des communications				
protection contre rejeu	oui	oui	oui	oui
protection contre le DoS	cookie	cookie	cookie	cookie
chiffrement métadonnées	oui	oui	oui	oui
agilité cryptographique	oui	oui	oui	oui
Éléments de sécurité				
crypto. asymétrique raisonnable sur le client	**non**	oui (communication avec le cache)	oui	oui
autorisation explicite	**non**	**non**	**non**	**non**
granularité d'autorisation	**SR**	**SR**	ressource	ressource
gestion automatique des clés	oui	oui	oui	oui
contexte par client (SR)	**oui**	**oui**	non	non
Contraintes				
nb relations entre domaines	8	8	8	8
horloge requise (SR)	oui *	non	non	non
service de cache de données	**non**	oui	oui	oui
surcharge taille de messages	**grande**	**grande**	faible	faible
sur le réseau contraint	6LoWPAN, UDP, DTLS, CoAP	6LoWPAN, UDP, DTLS, CoAP	DTLS, CoAP	CASAN-S
meilleur cas (octets)	40 : 3 + 4 + 29 + 4	40 : 3 + 4 + 29 + 4	29 + 4	17
pire cas (octets)	59 : 20 + 6 + 29 + 4	59 : 20 + 6 + 29 + 4	29 + 4	17
taille de la charge utile (octets)	45-64[1]	45-64[1]	71-85 (@ MAC courtes)	87-101 (@ MAC courtes)
nb msg démarrage du réseau (SR)	4 (IPv6)	4 (IPv6)	13 (association CASAN)	5 (association CASAN-S)
nb msg 1ère connexion (SR)	14 ** (10)	14 ** (10)	0 (connexion sur le maître)	0 (connexion sur le maître)
nb msg 1ère connexion (C)	14 ** (10)	14 ** (10)	14 ** (10)	14 ** (10)
nb msg à chaque requête (SR)	2	0 - 2	0 (cache) - 2	0 (cache) - 2
nb msg à chaque requête (C)	2	2	2	2
connexions maintenues (SR)	0	1 (avec le cache)	1 (avec le maître)	1 (avec le maître)
influence du nb de msg (SR)	**dépend du nb de clients et fréquence des requêtes**	faible (cache)	faible (cache)	faible (cache)
Aspects généraux				
protocoles utilisés	DTLS + CoAP	DTLS + CoAP	DTLS + CASAN	CASAN-S
arch avec app spécifique	non	non	non	non
découverte de ressources	**non**	**non**	oui	oui
services réseau requis sur SR	DHCP ou autoconf. IP, DNS	DHCP ou autoconf. IP, DNS	-	-
mise à l'échelle	**limitée**	**limitée**	bonne (agrégation des ressources sur le maître, chaînage des maîtres, maîtres multiples)	bonne (agrégation des ressources sur le maître, chaînage des maîtres, maîtres multiples)

TABLE 9.1 – Comparaison des architectures A-DTLS, CASAN et CASAN-S sur des aspects de sécurité et de contraintes. La couleur verte indique un avantage, la couleur rouge indique un inconvénient (ou alors le résultat dépend du déploiement), la couleur noire indique un résultat mitigé. * change au scénario 2. ** plus les paquets pour échanger les certificats.

4. nombre de messages au démarrage du réseau sur le serveur de ressources : au démarrage des architectures A-DTLS (avec et sans cache), le serveur de ressources doit récupérer une adresse IP, il utilise donc l'auto-configuration IPv6 qui nécessite 4 messages. L'architecture CASAN nécessite 13 messages pour l'association du maître et de son esclave (DTLS puis les messages CASAN). Enfin, l'architecture CASAN-S n'implique que les 5 messages d'association décrits dans la section précédente.

La *cryptographie asymétrique* n'est pas raisonnable dans l'architecture A-DTLS sans cache à cause du temps de calcul sur les serveurs de ressources. En revanche, l'authentification par clés asymétriques (ou avec des certificats) permet d'alléger le coût de maintenance, c'est-à-dire qu'il n'est pas nécessaire d'ajouter, de supprimer ou de renouveler des clés d'authentification pour chaque client sur chaque nœud qu'il contacte, un compromis est donc à faire suivant les besoins de déploiement. Dans tous les cas, A-DTLS avec cache, CASAN et CASAN-S ne nécessitent pas de cryptographie asymétrique sur les serveurs de ressources puisque la présence du maître est assurée.

Concernant la *granularité de l'autorisation*, A-DTLS (avec ou sans cache) assure que le serveur de ressources n'accepte que des requêtes venant de clients connus, puisque dans le cas contraire la connexion ne peut être établie. Tout client accepté a accès à toutes les ressources sur le serveur de ressources : il n'est pas prévu dans le mécanisme DTLS de gérer d'autorisation plus finement. CoAP permet de refuser l'accès à une ressource, mais dans ce cas cette gestion est faite sur chaque nœud, ou au niveau du cache de données, pas au niveau du protocole de communication sécurisée. CASAN et CASAN-S centralisent toute la gestion du contrôle d'accès sur le maître, avec une gestion fine jusqu'à la ressource, allégeant les serveurs de ressources de cette tâche.

Le nombre de relations entre les domaines est le même pour toutes les architectures, puisque le client se connecte avec le mécanisme DTLS dans tous les cas : sur le serveur de ressources ou le cache de données dans A-DTLS, et sur le maître dans CASAN et CASAN-S.

Le *nombre de messages à la première connexion* sur le serveur de ressources et sur le client diffèrent entre les architectures A-DTLS et CASAN. L'architecture A-DTLS nécessite au minimum 10 messages échangés entre le client et le serveur de ressources avec des clés pré-configurées, et jusqu'à 14 messages avec des certificats. CASAN et CASAN-S ne nécessitent que l'association de l'esclave au maître au démarrage du réseau, il n'y a pas de communication directe entre le client et l'esclave.

Le critère *influence de messages sur le serveur de ressources* indique de quoi dépend le nombre de messages sur le serveur de ressources. Dans l'architecture A-DTLS, le nombre de messages échangés avec le serveur de ressources dépend du nombre de clients et de leur fréquence de requête. Cette influence est limitée lorsqu'un cache de données est utilisé. Dans l'architecture A-DTLS avec un cache, le serveur de ressources se connecte au cache et la connexion reste active entre les requêtes.

Pour les *protocoles utilisés dans le domaine contraint*, A-DTLS nécessite une pile de communication complète (réseau, transport, sécurité, application), CASAN-S ne requiert que CoAP. De ce fait, CASAN-S est plus simple, ce qui implique potentiellement moins d'erreurs et donc diminue le coût de maintenance. Enfin, CASAN-S réduit les en-têtes des messages, permettant de partager plus de données par paquet.

Finalement, la *découverte de ressources* fournie directement par l'architecture permet d'économiser du temps et de l'énergie. 6LoWPAN associé à CoAP permet au client de découvrir les ressources, mais cela nécessite un mécanisme de multicast, qui peut ne pas être permis dans le réseau contraint, et qui est géré actuellement sans sécurité. CASAN-S a un mécanisme intégré pour la découverte de ressources, ne nécessitant aucun message supplémentaire dans le réseau contraint : chaque serveur de ressources envoie sa liste de ressources au maître une seule fois au démarrage du nœud.

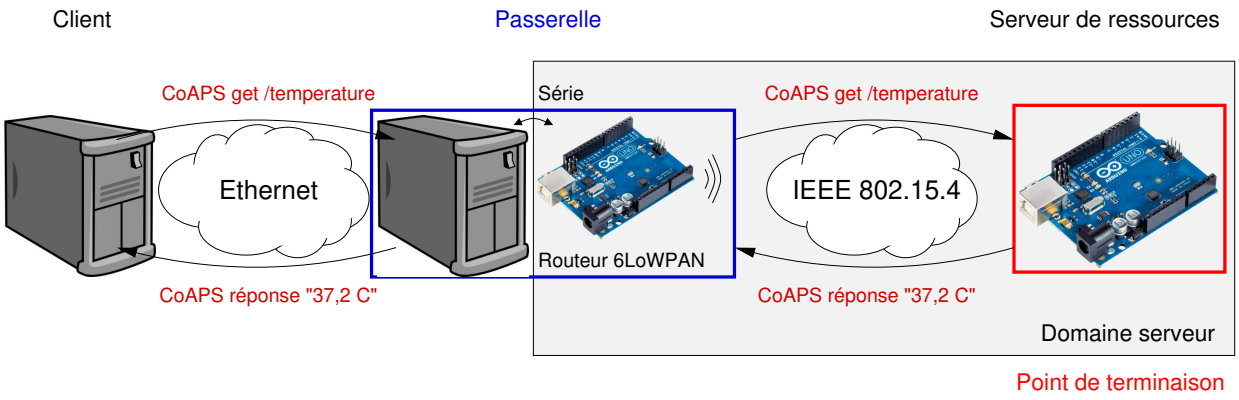


FIGURE 9.1 – Dispositif expérimental de l'architecture de référence (A-DTLS).

9.2 Mesure de l'influence

9.2.1 Architectures étudiées

Les architectures utilisées dans nos expérimentations sont A-DTLS pour l'architecture de référence, et CASAN-S pour l'architecture maître-esclave. A-DTLS est simple et très représentative d'une architecture de référence ; dans la philosophie initiale de l'Internet, l'intelligence est uniquement aux extrémités. Elle utilise les protocoles standards préconisés par l'IETF pour la communication entre un client et un serveur de ressources, à savoir DTLS pour la communication sécurisée et CoAP pour la couche applicative. CASAN-S est une architecture maître-esclave, qui exploite pleinement les avantages de ce type d'architecture. Par exemple, la pile de communication sur le réseau contraint est indépendante de celle du client (pile protocolaire minimaliste, pas de connexion du client aux serveurs de ressources, etc.).

9.2.2 Mesures

Nos mesures portent dans un premier temps sur le temps de démarrage d'un réseau, pour savoir en combien de temps il devient stable. Nous quantifions ainsi l'impact d'une insertion du serveur de ressources dans le réseau, ce qui correspond à l'association pour CASAN-S ou à l'obtention des informations IP pour A-DTLS. Dans un second temps, nous mesurons le temps de requête des capteurs. Nous quantifions ainsi l'impact d'une requête d'un client sur le réseau, ce qui correspond à l'établissement d'une connexion sécurisée entre le client et les nœuds contraints pour A-DTLS, et entre le client et le maître pour CASAN-S.

9.3 Dispositif expérimental

Afin de mesurer les performances des deux architectures, nous avons mis en place un dispositif expérimental dédié aux mesures des temps nécessaires pour démarrer les architectures ainsi que pour effectuer une requête.

Le matériel utilisé pour les nœuds contraints est un Zigduino [97] composé d'un micro-contrôleur 8 bits ATmega128RFA1 intégrant un composant de communication IEEE 802.15.4 (2006), 16 Ko de RAM, et 128 Ko de mémoire flash pour stocker le code du programme.

9.3.1 Architecture de référence (A-DTLS)

La figure 9.1 montre le dispositif expérimental de l'architecture de référence, dans un réseau IEEE 802.15.4 (2006), avec RPL et IPv6 via 6LoWPAN :

— Le client est un système Linux non contraint avec une application CoAPS, connecté au routeur par une liaison série à 57600 bauds [1] ;

— Le routeur est un Zigduino avec le logiciel Contiki OS [98], il gère un réseau 6LoWPAN en IEEE 802.15.4 [2] ;

— Le serveur de ressources est un autre Zigduino. Il utilise la bibliothèque logicielle PicoIPv6 [99] comme implémentation de 6LoWPAN et RPL, conjointement avec la bibliothèque de communication 802.15.4 utilisée également pour l'architecture maître-esclave. Nous utilisons le concept de contextes 6LoWPAN vu au chapitre 2 (page 10) pour diminuer la taille des en-têtes IPv6 [3].

9.3.2 Architecture maître-esclave (CASAN-S)

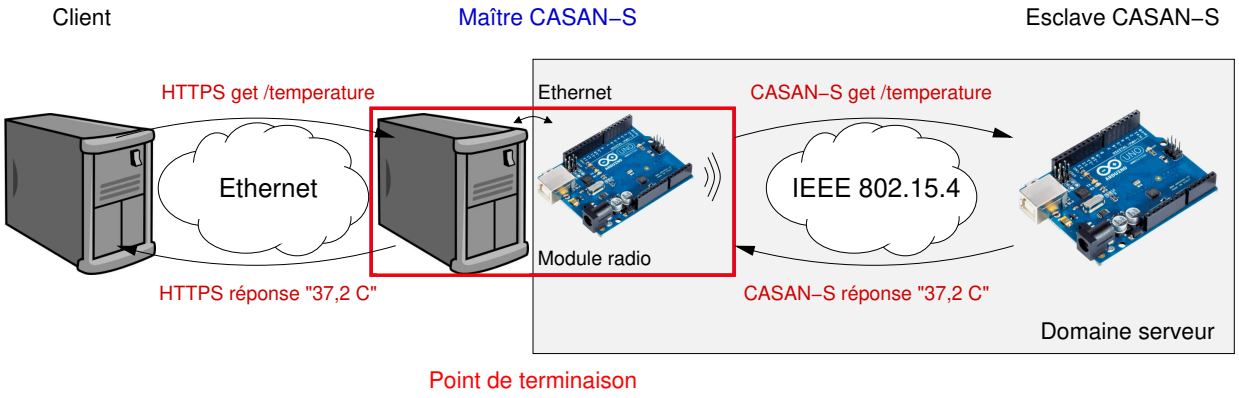


FIGURE 9.2 – Dispositif expérimental de l'architecture maître-esclave (CASAN-S) (avec module IEEE 802.15.4 par liaison Ethernet).

La figure 9.2 montre le dispositif expérimental de l'architecture maître-esclave :

— le client est un programme lancé sur un hôte Linux ;

— le maître est également un programme lancé sur un hôte Linux, relié à ses esclaves par un module IEEE 802.15.4 ;

— le module IEEE 802.15.4 est un Zigduino relié en Ethernet au maître ;

— le serveur de ressources est un Zigduino, dont la bibliothèque de communication 802.15.4 est identique à celle utilisée pour l'architecture de référence.

Le lecteur pourra noter que l'interface entre le client et le réseau 802.15.4 (routeur 6LoWPAN pour A-DTLS ou maître pour CASAN-S) est assurée par une technologie différente entre A-DTLS (liaison série et Serial Line IP) et CASAN-S (Ethernet). Cela présente un biais et nous le quantifierons en calculant et mesurant le temps nécessaire à l'échange de données sur cette liaison série. De plus, nous neutralisons ce biais pour certaines mesures en utilisant un module spécifique (Stick Digi XBee) avec une liaison série à 57600 bauds au maître.

1. La liaison série est établie avec le programme `tunslip6`, fourni par Contiki.
2. Le logiciel Contiki est utilisé en version 3.1, avec le firmware `examples/ipv6/rpl-border-router`. Le routeur est configuré pour retransmettre les paquets jusqu'à 5 fois sur le réseau en cas de perte.
3. Dans notre environnement, l'adresse est incluse dans le message, nous pourrions donc avoir des réponses plus courtes de 8 octets que dans nos mesures.

9.4 Expérimentations

Dans cette section, nous décrivons précisément les éléments que nous mesurons, les résultats faisant l'objet des sections suivantes.

9.4.1 Scénario n°1 : insertion dans le réseau

Dans ce premier scénario, nous démarrons tous les nœuds d'un réseau en même temps et analysons le temps nécessaire pour que chaque nœud soit opérationnel. Nous mesurons donc le temps entre l'envoi du premier message d'insertion et la réception du dernier, ce qui correspond à un état stabilisé du réseau. Les architectures effectuent des opérations différentes lors de l'insertion : dans l'architecture maître-esclave, les appareils contraints (esclaves) se connectent à un nœud central (maître) au démarrage du réseau et créent une communication sécurisée entre les deux, ce qui implique des opérations cryptographiques. Dans l'architecture de référence, l'insertion d'un nœud dans le réseau implique de lui attribuer des adresses et éventuellement mettre à jour la table de routage de la passerelle entre le réseau contraint et le reste d'Internet.

9.4.1.1 Déroulement des messages dans l'architecture CASAN-S

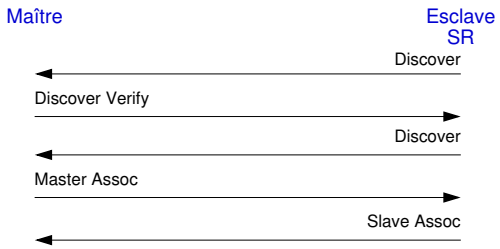


FIGURE 9.3 – Mesure du temps d'insertion dans un réseau CASAN-S.

La figure 9.3 montre le démarrage d'un réseau avec l'architecture CASAN-S. L'esclave découvre son maître (message « Discover ») qui lui répond en donnant un cookie (message « Discover Verify ») que l'esclave devra renvoyer (de nouveau un message « Discover »). Une fois que l'esclave a prouvé qu'il écoutait le réseau et qu'il ne se contentait pas de rejouer un scénario d'attaque, le maître lui envoie la preuve de sa possession de la clé secrète (et donc son identité) via le message « Master Assoc ». Enfin, l'esclave répond en donnant lui aussi la preuve de la possession de la clé secrète, ainsi que la liste chiffrée de ses ressources dans le message « Slave Assoc ». Ces messages constituent le démarrage d'un nœud CASAN-S. La durée mesurée correspond donc au temps entre l'envoi du message Discover et la réception du message Slave Assoc.

9.4.1.2 Déroulement des messages dans l'architecture A-DTLS

La figure 9.4 montre le démarrage d'un réseau avec l'architecture A-DTLS. Une différence notable avec CASAN-S est que les échanges se déroulent exclusivement entre les nœuds sur le réseau IEEE 802.15.4. Lorsqu'un nœud démarre dans le réseau A-DTLS, il envoie un message DIS (DODAG Information Sollicitation) pour demander un DIO (DODAG Information Object) auprès du routeur[4]. Le message DIO contient les informations liées au réseau, comme le préfixe. Le nœud construit ses

4. Le standard RPL [18] ne force pas les nœuds à envoyer un message de sollicitation, les nœuds pouvant attendre un message périodique envoyé en broadcast par le routeur. Cependant pour comparer les temps de démarrage avec ceux de CASAN-S nous avons choisi de forcer les nœuds à solliciter le routeur pour obtenir des temps les plus courts possibles sans délai aléatoire. Ce comportement décrit dans la RFC 6550 (section 18.2.1.1) qui stipule qu'un nœud peut demander des informations sur l'instance RPL en cours dès son démarrage via un message DIS.

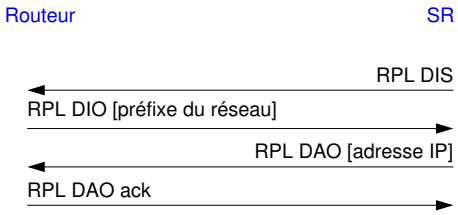


FIGURE 9.4 – Mesure du temps d'insertion dans un réseau A-DTLS.

adresses IPv6 locale et globale à partir de celui-ci. Puis, le nœud avertit le routeur de sa présence (et annonce son adresse IP) avec un message DAO (Destination Advertisment Object), ce qui permet au routeur de compléter sa table de routage. La durée mesurée est entre l'envoi du message DIS et l'envoi du message DAO.

9.4.1.3 Observation attendue

Nous devrions observer une durée de démarrage et de stabilisation du réseau plus rapide dans l'architecture A-DTLS. Cette architecture échange moins de messages lors du démarrage du réseau et n'implique pas de cryptographie.

9.4.2 Scénario n°2 : requête des nœuds

Dans notre seconde expérimentation, nous souhaitons mesurer la durée d'une requête, du premier au dernier message échangé. Cette durée couvre l'établissement de la connexion, la requête suivie de sa réponse, ainsi que la fin de la connexion (message d'alerte DTLS ou déconnexion TCP). Les requêtes sont envoyées en parallèle à tous les nœuds du réseau à partir d'un client.

Les architectures traitent différemment les requêtes. Une architecture de référence nécessite une communication sécurisée entre le client et le serveur de ressources, ce qui implique d'échanger des messages et d'authentifier le client sur un nœud contraint. À l'inverse, dans une architecture maître-esclave, le client ne se connecte pas au nœud contraint mais au maître, qui dispose déjà d'une communication sécurisée avec son esclave. Les nœuds contraints n'effectuent que des opérations de chiffrement et de déchiffrement, et l'authentification est réalisée par le maître.

9.4.2.1 Déroulement des messages dans l'architecture CASAN-S

La figure 9.5 montre une requête d'un client pour récupérer une donnée. Le client se connecte de manière sécurisée au maître avec une communication TLS, puis lui envoie une requête HTTP. Le maître traduit la requête HTTP en CASAN-S pour l'envoyer à l'esclave. Enfin, suite à la réponse du maître, le client termine la connexion. La durée mesurée correspond au temps entre le premier et le dernier message TCP entre le client et le maître.

9.4.2.2 Déroulement des messages dans l'architecture A-DTLS

La figure 9.6 montre une requête d'un client CoAPS au serveur de données. Les messages représentent une connexion DTLS, une requête CoAP sur DTLS puis une terminaison de session via un message *alerte* lancé par le client pour informer le serveur de ressources de la fin de communication. Tous les messages sont échangés entre le client et le serveur de ressources, donc via le réseau contraint.

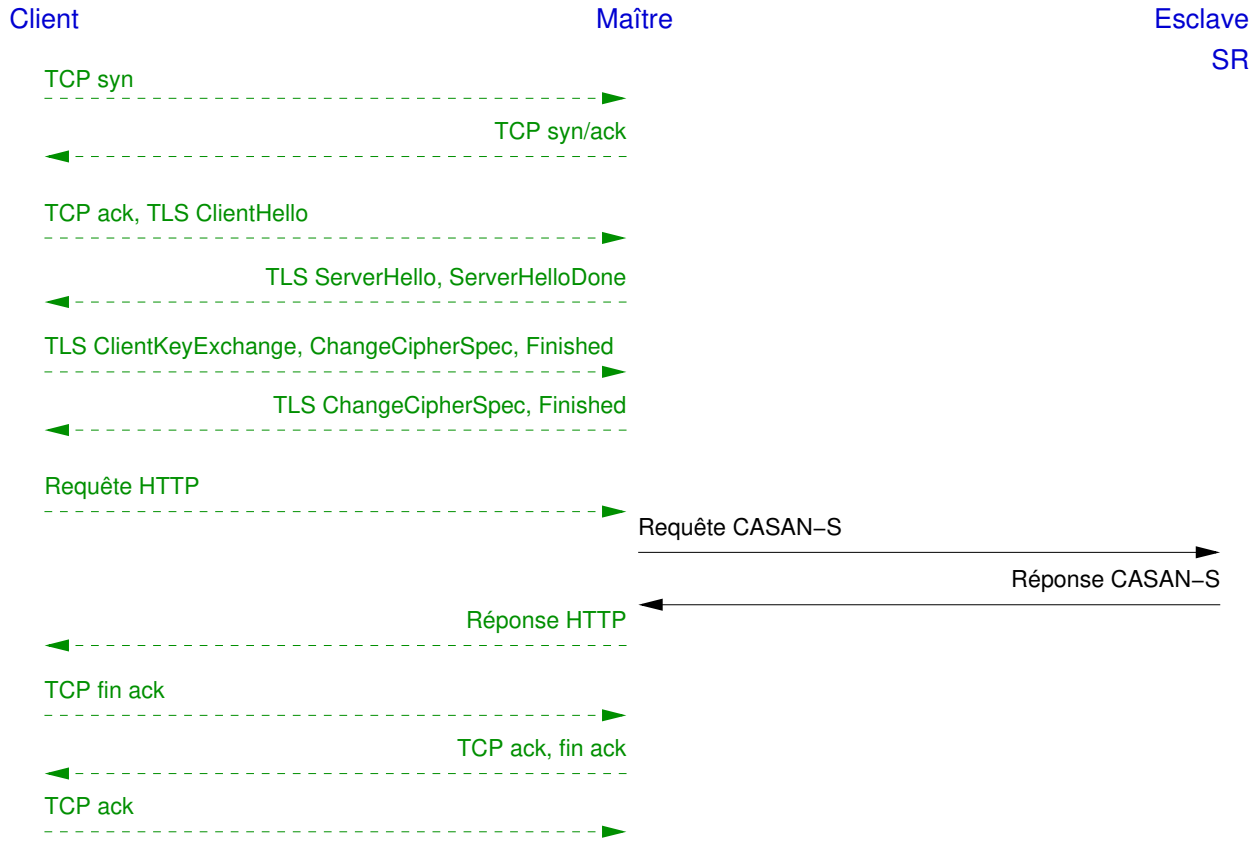


FIGURE 9.5 – Requête d'un nœud avec l'architecture CASAN-S. Les échanges en vert et traits discontinus se font sur le réseau non contraint.

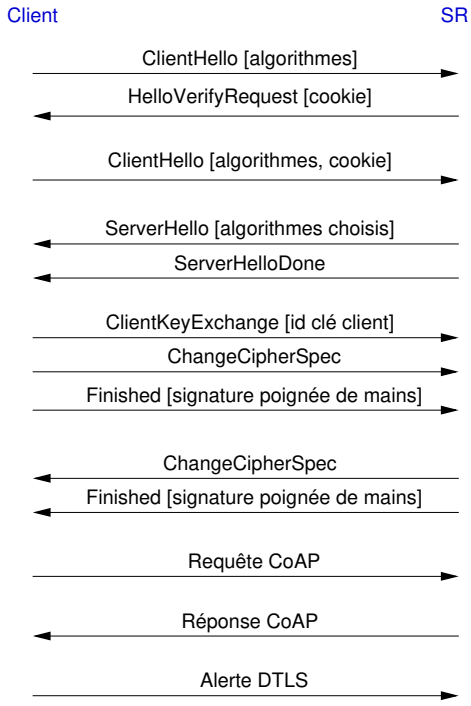


FIGURE 9.6 – Requête d'un nœud avec l'architecture A-DTLS.

9.4.2.3 Observation attendue

Nous devrions observer une durée de requête bien plus courte dans l'architecture CASAN-S pour deux raisons : premièrement, la requête à l'esclave puis sa réponse représentent les deux seuls messages sur le réseau contraint, ce qui signifie une faible attente avant l'accès au médium et une faible consommation de bande passante (et donc une meilleure mise à l'échelle). Deuxièmement, une requête n'implique pas une nouvelle connexion sur le nœud contraint, ce nœud n'effectue pas d'opérations d'authentification, il se charge uniquement du chiffrement et du déchiffrement des messages. À l'inverse, dans l'architecture A-DTLS, les nombreux messages d'une connexion sont envoyés sur le réseau contraint, induisant une latence à chaque message (c'est-à-dire un temps d'attente, un temps d'envoi et un temps de propagation). De plus, le serveur de ressources effectue l'authentification du client, imposant un temps de calcul cryptographique non négligeable.

9.4.3 Paramètres des expérimentations

Nous faisons varier la taille de notre réseau expérimental de 1 à 9 nœuds et nous effectuons 100 essais par configuration. L'augmentation du nombre de nœuds au delà de 9 dans le réseau échangeant des messages implique une augmentation du temps d'attente sur le réseau 802.15.4. Nous n'avons pas augmenté le nombre de nœuds car nous atteignons rapidement les limites des équipements, ce qui sera détaillé dans la section suivante.

Les mesures effectuées sont limitées dans le temps, pour que les durées d'attente avant une retransmission ne biaisent pas les résultats. Lors des mesures des temps de démarrage des architectures, nous limitons la durée d'une mesure à 5 secondes, donc seuls les temps de démarrage inférieurs à $5 - \delta$ secondes sont pris en compte (δ étant le temps de démarrage matériel du nœud, soit légèrement plus d'une seconde) [5]. Lors des mesures des durées de requêtes, toute requête n'étant pas terminée en moins de 10 secondes n'est pas prise en compte dans le calcul des durées moyennes.

L'architecture maître-esclave permet l'utilisation de tous les algorithmes cryptographiques et donc de simplifier la gestion du matériel cryptographique grâce au maître non contraint, ce que nous mettons en avant dans nos expérimentations ; le maître utilise un certificat, ainsi que des algorithmes asymétriques pour l'authentification (ECDSA) et l'échange de clés (ECDH). ECDSA permet au maître de ne disposer que d'une paire de clés pour l'authentification, et ECDH permet de générer un secret partagé avec chaque client à la connexion sans nécessiter d'en conserver. Cette simplification n'est pas possible dans l'architecture de référence qui nécessite l'utilisation d'algorithmes symétriques (AES 128 bits en CCM, avec clé pré-configurée pour l'authentification) forçant les serveurs de ressources à conserver un secret par client. Par conséquent, le choix des algorithmes représente un scénario réaliste pour chacune des architectures (choix en fonction des capacités des matériels) et est en défaveur de l'architecture maître-esclave qui utilise des algorithmes plus complexes. Nous verrons cependant que la durée d'authentification est plus courte sur un nœud non contraint que sur un nœud contraint, peu importe le type d'algorithme utilisé.

9.4.4 Comparaison avec des valeurs théoriques

Pour effectuer dans la section suivante une étude analytique des résultats, nous calculons les valeurs théoriques des durées d'échanges sur le réseau pour les deux scénarii.

Nous aurons besoin de calculer les temps de transfert et d'attente des différents messages sur le réseau (et sur une liaison série). Pour cela, nous prendrons comme base le temps de transfert sur le réseau 802.15.4 qui est de 31,25 octets par milliseconde [6].

5. Nous obtenons expérimentalement une approximation de la durée de démarrage de l'appareil (δ) en diminuant progressivement la durée d'un essai. Par exemple, nous passons la durée d'un essai à 3, puis 2 puis une seconde : le matériel n'a pas le temps de démarrer avec une durée d'essai d'une seconde.

6. La vitesse de communication 802.15.4 dans la plage de fréquences des 2,4 GHz est de 62500 symboles par seconde, et un octet est représenté par 2 symboles soit 31,25 octets par milliseconde.

Nous considérons uniquement le temps moyen d'attente du premier essai d'envoi d'un message lors des calculs des temps de transmission théoriques, ce qui représente une approximation basse du temps d'attente en pratique [7]. Nous prendrons en compte le temps d'attente 802.15.4 de l'algorithme CSMA-CA est de 1,12 ms en moyenne, voir section 2.2.3 (page 8).

La vitesse de communication d'une liaison série est comprise entre 9600 et 57600 bauds sur nos appareils (soit de 960 à 5760 octets/s, voir annexe D).

Les tailles des messages lors de l'insertion d'un nœud ou d'une requête dans le réseau sont fournies dans l'annexe B (page 123).

9.4.5 Conclusion

Nous avons vu dans cette section les scénarii que nous allons étudier : le démarrage d'un réseau (ce qui correspond à l'insertion des nœuds dans l'architecture) et une requête d'un client sur un nœud. La taille du réseau varie de 1 à 9 nœuds dans un réseau IEEE 802.15.4. Chaque configuration est testée sur 100 essais. Enfin, pour établir des résultats théoriques nous utilisons les durées calculées de transfert et d'attente sur des liens, ainsi qu'une mesure des durées des opérations cryptographiques.

9.5 Résultats : durée de démarrage du réseau

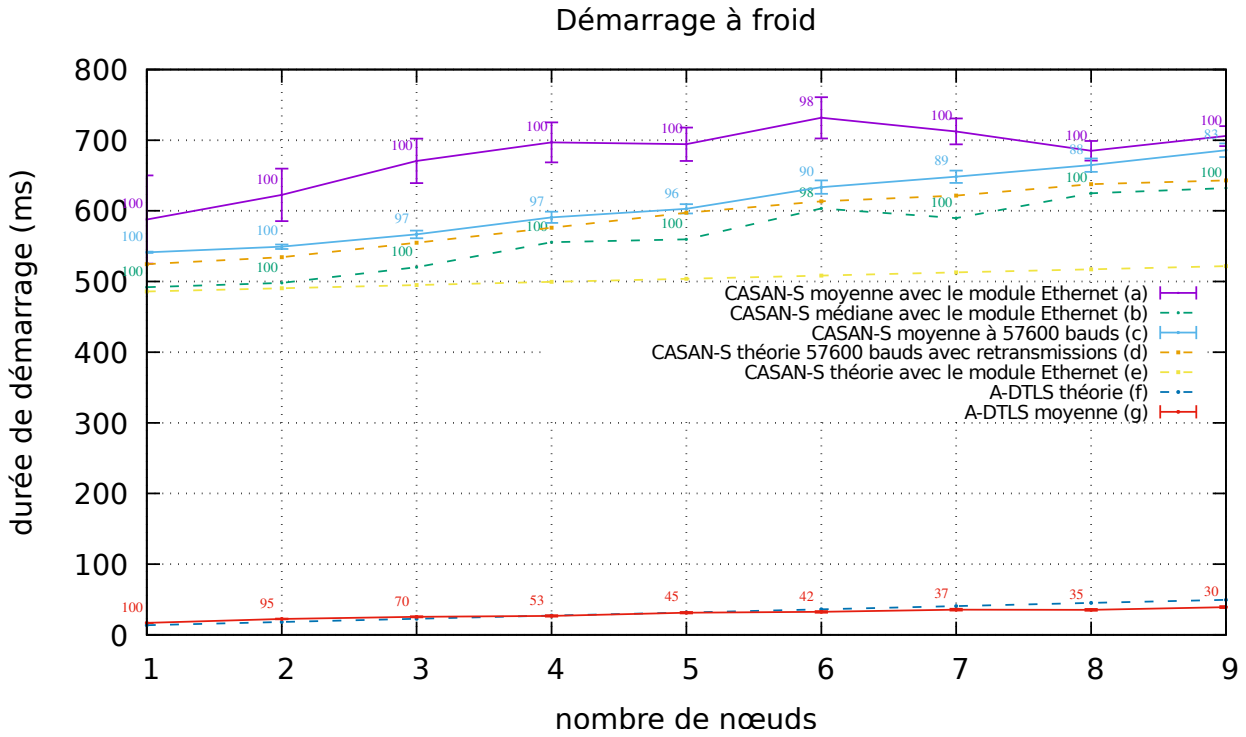


FIGURE 9.7 – Mesure du temps de démarrage d'un réseau dans les architectures A-DTLS et CASAN-S (avec module Ethernet et liaison série).

Dans cette section, nous allons voir que le démarrage d'une architecture A-DTLS est plus rapide que CASAN-S car la durée des opérations cryptographiques dans CASAN-S retarde la stabilisation du réseau, tout en restant en dessous de 500 ms pour l'insertion d'un nœud dans l'architecture. Nous

7. Calculer le temps d'attente théorique lors de plusieurs échanges simultanés est un exercice difficile : la perte d'un message implique l'arrêt de l'échange, et les durées d'attente sont négligeables face au temps pris par les opérations cryptographiques.

présentons les résultats obtenus pour le premier scénario, puis nous présentons et expliquons ces résultats. Un autre enseignement de ces mesures est que l'insertion avec A-DTLS s'effondre dès que nous dépassons 2 nœuds simultanés, ce qui a motivé notre choix de ne pas augmenter davantage le nombre de nœuds dans notre dispositif expérimental.

La figure 9.7 présente les courbes combinées des temps moyens de démarrage dans les architectures A-DTLS et CASAN-S. Les courbes pleines représentent les moyennes des mesures des durées de démarrage (avec module Ethernet ou liaison série pour CASAN-S). Les traits discontinus représentent les médianes et les courbes théoriques, calculées à partir des tailles des messages et de la durée théorique des échanges sur IEEE 802.15.4 (et sur liaison série). Ces courbes incluent également les pertes et temps de retransmission (voir annexe C). Le taux de démarrages réussis est indiqué sur chaque point des courbes de mesures. Nous pouvons voir une chute importante du taux de complétion de démarrages dans l'architecture A-DTLS (courbe (g), qui passe de 95 % pour 2 nœuds à 53 % pour 4 nœuds).

9.5.1 La mesure pour A-DTLS

Cette courbe n'augmente pas comme la courbe théorique : en effet, lors d'une perte de paquet non liée au réseau (problème à cause d'une mémoire tampon insuffisante, voir annexe C), le nœud attend plusieurs secondes avant d'envoyer à nouveau le message. La connexion n'est alors pas terminée à temps et n'est pas prise en compte dans la moyenne. Les messages de connexion suivants ne sont pas envoyés, le réseau est moins congestionné, et finalement les nœuds n'ayant pas perdu de paquet terminent plus rapidement leur connexion au réseau. Par conséquent, un essai avec un faible taux de réussite peut avoir un temps moyen plus faible qu'en théorie si seuls les premiers nœuds à terminer leur connexion sont pris en compte. Si les pertes surviennent au début de la connexion, l'envoi des autres messages est retardé au-delà de la durée d'un essai : c'est ce qui arrive à la courbe A-DTLS avec plus de 5 nœuds. De plus, le serveur de ressources attend 8 secondes avant de retransmettre un message en cas de non réponse, donc toute perte entraîne un échec de la connexion dans notre expérimentation.

9.5.2 Les mesures pour CASAN-S

Les temps d'insertion pour CASAN-S sont plus importants que A-DTLS, mais CASAN-S n'est pas sujet à l'effondrement constaté par A-DTLS du taux de complétion.

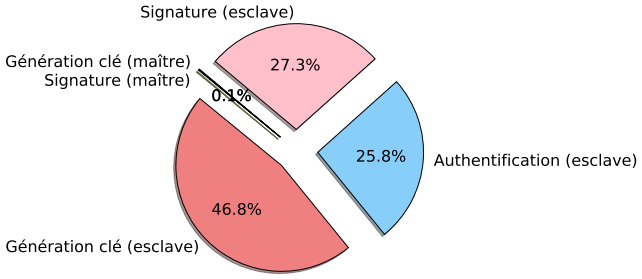
Nous avons mesuré les durées moyennes de démarrage de l'architecture CASAN-S avec un module Ethernet (figure 9.7, courbe a) et avec un module avec liaison série (courbe c) reliant le maître au réseau contraint. La médiane pour Ethernet est également indiquée (courbe b), ainsi que les valeurs théoriques pour la liaison série et prenant en compte les retransmissions (courbe d) et les valeurs théoriques pour Ethernet (courbe e).

Nous observons une courbe de moyenne Ethernet qui montre des temps moins bons que la courbe de moyenne avec liaison série malgré une bande passante beaucoup plus importante. Cette différence s'explique par un taux de pertes plus important avec la liaison série. Les durées de retransmission lors d'une perte sur CASAN-S sont de 150 à 450 ms, ce qui permet à CASAN-S de garder un bon taux de complétion. Cette durée avant retransmission permet de terminer le démarrage dans l'intervalle de temps imparti pour la mesure, c'est-à-dire en moins de 5 secondes (voir les paramètres d'expérimentation section 9.4.3 page 101). La courbe médiane b des durées de démarrage avec le module Ethernet montre que la moitié des démarrages se terminent en un temps proche de la valeur théorique lors des essais à un et deux nœuds. Au-delà, la courbe n'augmente plus de manière linéaire, indiquant la présence de pertes suivies de retransmissions, ce qui est d'autant plus marqué lors de la comparaison avec la moyenne (courbe a) : la médiane se rapproche de la moyenne, de moins en moins de nœuds terminent leur démarrage sans perte. Les pertes peuvent être reportées au delà de l'intervalle de mesure ; le taux de complétion des démarrages est alors inférieur mais la moyenne est meilleure.

Nous observons également un taux de complétion bien meilleur avec le Zigduino Ethernet qu'avec le Stick Digi via la liaison série (où ce taux descend jusqu'à 83 %), ce qui s'explique par un plus faible taux de pertes (voir annexe C).

Opération	Durée (ms)	Écart type (ms)	p-valeur
Génération de clé (maître)	0,540	0,742	$4{,}1 \times 10^{-11}$
Génération de signature (maître)	0,316	0,413	$6{,}3 \times 10^{-12}$
Génération de clé (esclave)	218	0,216	$2{,}2 \times 10^{-16}$
Authentification du maître	120	0	-
Génération de signature (esclave)	127	0,216	$2{,}2 \times 10^{-16}$

(A) Mesures des durées des fonctions cryptographiques au démarrage d'un nœud.



(B) Proportion des durées cryptographiques au démarrage d'un nœud CASAN-S.

FIGURE 9.8 – Opérations cryptographiques lors d'une requête dans le réseau CASAN-S. La durée de génération de la signature sur l'esclave comprend également la génération et la mise en file d'attente du message Slave Assoc.

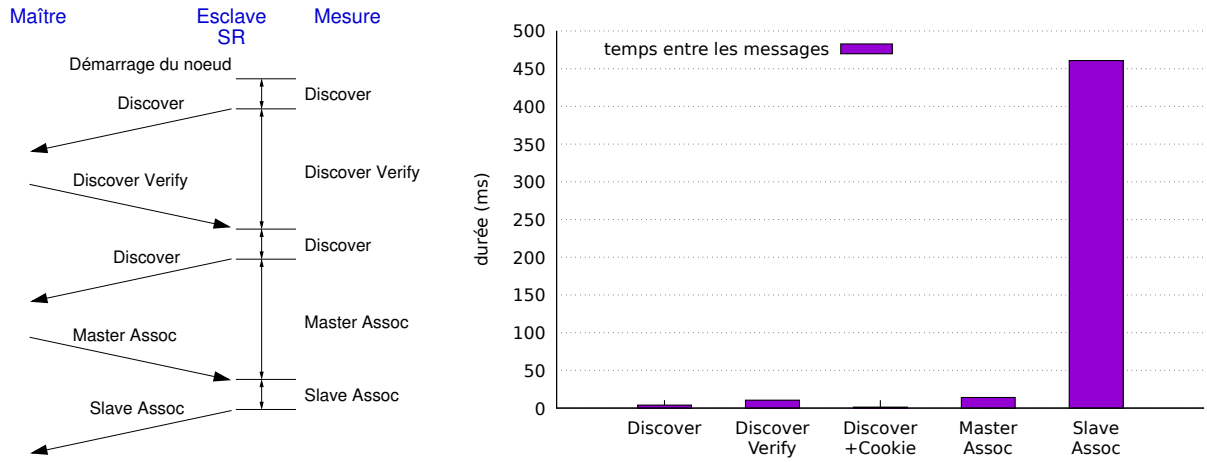
9.5.3 Comparaison des architectures sur les durées d'insertion

Contrairement à l'architecture A-DTLS, la durée des échanges avec l'architecture CASAN-S ne correspond pas simplement au temps d'attente et de transfert sur le lien 802.15.4 : la différence majeure entre ces deux architectures réside dans les opérations cryptographiques effectuées. En effet, dans l'architecture maître-esclave, l'esclave a une communication sécurisée avec le maître et, lors de son démarrage, ils construisent des clés cryptographiques puis ils s'authentifient. La figure 9.8a montre les durées cryptographiques, mesurées sur 100 essais et la figure 9.8b illustre la proportion de ces durées. Ce qui est appelé *signature* ici est un condensat chiffré représentant les données de la poignée de mains, nous ne parlons pas ici d'une signature réalisée avec un algorithme asymétrique. La vitesse mesurée des opérations varie davantage sur le maître que sur l'esclave (voir p-valeur plus importante sur la figure 9.8a), du fait de l'exécution des opérations sur un système généraliste avec de multiples programmes en concurrence pour l'accès au CPU. Cependant, les durées de ces opérations sont négligeables en comparaison des durées des opérations cryptographiques sur l'esclave, qui sont de plusieurs ordres de grandeur supérieures.

9.5.4 Impact de la cryptographie lors de l'insertion d'un nœud dans CASAN-S

Les opérations cryptographiques sont identifiables dans les délais des messages, en analysant les temps d'attente entre les différents messages. La figure 9.9a montre les opérations mesurées sur l'esclave, et la figure 9.9b illustre les résultats. La quasi-totalité du temps de démarrage se situe avant l'envoi du message « Slave Assoc », lorsque l'esclave doit effectuer toutes ses opérations cryptographiques.

Le premier message est envoyé après un temps de démarrage logiciel de quelques millisecondes (après le démarrage matériel). Le message « Discover Verify » est reçu le temps d'un aller-retour sur le réseau IEEE 802.15.4, aucun calcul n'est effectué. De même, le message suivant est envoyé rapidement par l'esclave (moins de 2 ms) puisque la seule opération à effectuer est une copie de mémoire



(A) Mesures effectuées.

(B) Temps observés entre les messages.

FIGURE 9.9 – Temps entre les messages sur le serveur de ressources dans l'architecture CASAN-S.

(récupération du cookie du message précédent). Ensuite le message Master Assoc, qui comporte une donnée d'authentification du maître, est reçu (presque) le temps d'un aller-retour car les opérations cryptographiques sont faites sur le maître. Enfin, la quasi-totalité du temps de démarrage du nœud est liée au traitement du message Master Assoc et à l'envoi du message Slave Assoc. À la réception du message Master Assoc, l'esclave crée des clés de chiffrement, vérifie la signature contenue dans le message et génère une signature pour le message Slave Assoc. Cette représentation met en évidence les opérations cryptographiques, et que les durées d'échange et d'attente sur le lien sont un ordre de grandeur inférieures : l'échange d'un message se fait en moins de 10 ms et le cumul des durées des opérations cryptographiques mesuré sur l'esclave est de 470 ms.

9.5.5 Conclusion

Dans cette section, nous avons observé que l'architecture A-DTLS est plus rapide à se stabiliser lors du démarrage que dans CASAN-S. Dans l'architecture A-DTLS, la durée de stabilisation des nœuds s'explique par les durées d'attente et de transfert des messages. Dans l'architecture CASAN-S, une connexion authentifiée s'opère entre le maître et ses esclaves, la durée des opérations cryptographiques est alors bien plus importante que la durée des échanges, ce qui marque la principale différence des deux architectures lors du démarrage d'un réseau.

Cette connexion sécurisée dans l'architecture CASAN-S permettra des temps de réponse rapides lors de requêtes, ce que nous verrons dans la section suivante.

9.6 Résultats : durée d'une requête

Dans cette section, nous allons voir que les requêtes dans une architecture CASAN-S sont bien plus rapides que dans une architecture A-DTLS, malgré l'utilisation d'algorithmes asymétriques dans CASAN-S pour l'authentification des clients et le partage de clés permettant de simplifier la gestion du matériel cryptographique. Nous présentons puis expliquons les résultats obtenus lors de la mesure des durées de requête dans les architecture A-DTLS et CASAN-S.

La figure 9.10 montre les temps de requête mesurés avec ces deux architectures. Le temps de requête d'un capteur dans l'architecture CASAN-S est plusieurs ordres de grandeur inférieur au temps de requête dans l'architecture A-DTLS. De plus, les requêtes dans CASAN-S conservent un taux de complétion de 100 %, contrairement à l'architecture A-DTLS qui chute d'un taux de complétion de 93 % à 2 nœuds, à seulement 17 % à 4 nœuds. La suite de cette section explique ces résultats.

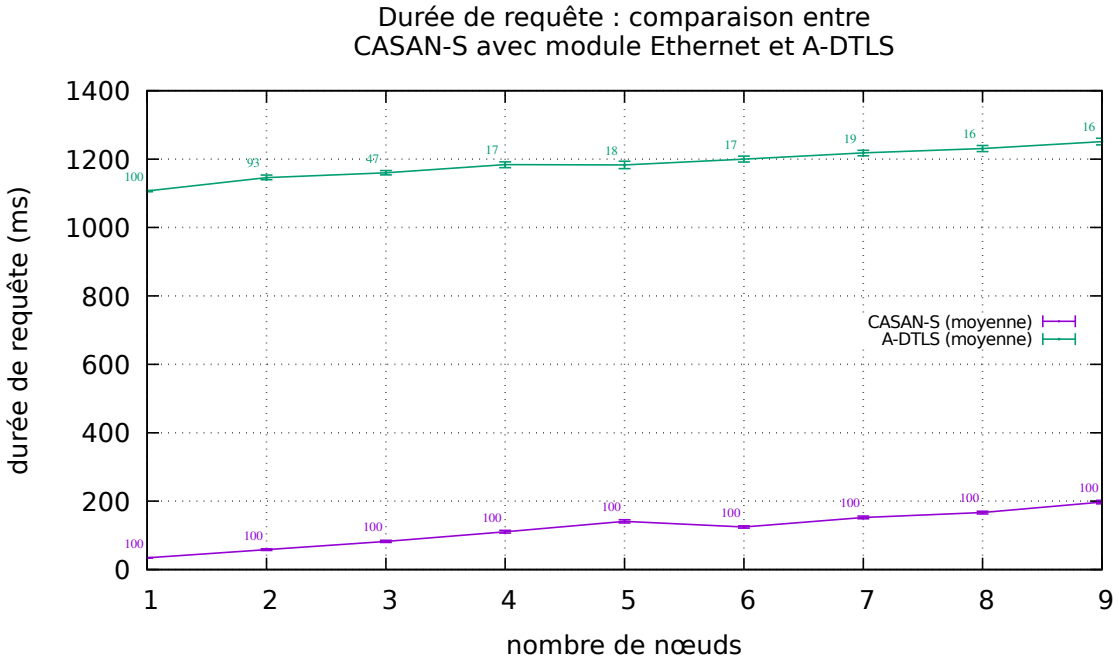


FIGURE 9.10 – Mesure du temps de requête dans les architectures A-DTLS et CASAN-S. La liaison série entre le routeur et le client A-DTLS est configurée à 57600 bauds. Le taux de requêtes réussies est indiqué au dessus de chaque point.

9.6.1 Pertes sur la liaison série

Des pertes surviennent sur la liaison série lorsque de nombreux échanges se font sur le réseau 802.15.4 (voir annexe C), ce qui indique un problème de mémoire tampon insuffisante pour stocker les messages en attente de transfert sur la liaison série. Par conséquent, le taux d'échec de connexion important : le taux de complétion chute à seulement 16 % lors de requêtes dans l'architecture de référence. Le taux de complétion des requêtes étant encore plus faible que celui des démarrages de l'architecture A-DTLS, il a de nouveau été décidé de ne pas augmenter davantage le nombre de nœuds dans notre expérimentation.

9.6.2 Influence de la connexion HTTPS sur CASAN-S

Nous pouvons voir que les requêtes dans l'architecture maître-esclave sont jusqu'à plusieurs dizaines de fois plus rapides qu'avec l'architecture de référence, malgré un choix d'algorithmes fortement en faveur de l'architecture de référence (voir les paramètres d'expérimentation section 9.4.3 page 101). Cet écart diminue avec le nombre de nœuds, et à 9 nœuds l'architecture maître-esclave n'est plus que 6 fois plus rapide pour effectuer ses requêtes, tout en gardant un taux de complétion de 100 % (alors qu'il chute à 16 % avec A-DTLS) car les pertes impliquent des retransmissions au-delà de l'intervalle de mesure (qui est de 10 secondes pour une requête) ; la moyenne des durées de requêtes n'augmente donc pas autant que prévu car elle n'inclut que les requêtes complétées à temps (16 % des requêtes à 9 nœuds). Le temps de requête augmente davantage en fonction du nombre de nœuds que du temps d'échange sur le réseau contraint : cela s'explique par le manque de parallélisation des calculs de cryptographie asymétrique sur le maître CASAN-S.

Les durées des opérations cryptographiques, d'attente et de transfert sur le réseau IEEE 802.15.4 seront analysées et quantifiées par la suite, ainsi que les durées de transfert sur la liaison série pour l'architecture A-DTLS.

9.6.3 Opérations cryptographiques sur matériel non contraint

Opération cryptographique	Durée ou vitesse
ECDH 256 bit (nistp256)	0,4 ms
AES 128 GCM (blocs de 16 octets)	1 809 856 octets / ms
SHA 256 (blocs de 16 octets)	720 032 octets / ms
RSA 4096 bits, signature	9,521 ms
RSA 4096 bits, vérification	0,246 ms

TABLE 9.2 – Durées des opérations cryptographiques sur matériel non contraint.

Pour mieux analyser les mesures des requêtes, nous avons quantifié la durée des opérations cryptographiques sur le matériel non contraint utilisé en tant que client. Le tableau 9.2 récapitule la durée mesurée des opérations cryptographiques impliquées dans une requête CASAN-S[8]. En dehors de la signature (9,5 ms) et de la négociation de la clé de chiffrement (0,4 ms), les autres opérations ont une durée négligeable. Le client de l'architecture A-DTLS n'utilise que de la cryptographie symétrique, dont la durée des calculs est négligeable sur un matériel non contraint.

Opération	Durée (ms)	Obtention du résultat
établissement de la connexion HTTPS du client au maître	10,167	mesuré
durée de traitement d'une requête	6,63	mesuré sur 100 essais (écart type = 0,56 ms)
échange sur réseau 802.15.4	2,46	$temps_{\text{transfert}} = \frac{taille_{\text{données}}}{vitesse} = (67 + 6 \times 2) \times 0,03125$
attente sur le réseau 802.15.4	2,24	2×1,12
Total calculé	21,49	somme des durées précédentes
Total observé	34,13	mesuré sur 100 essais (écart type = 0,915 ms)

TABLE 9.3 – Durées calculées et observées pour une requête sur un nœud dans l'architecture CASAN-S avec une liaison Ethernet entre le maître et un nœud Zigduino faisant office d'émetteur et de récepteur. Le client est simulé sur le maître.

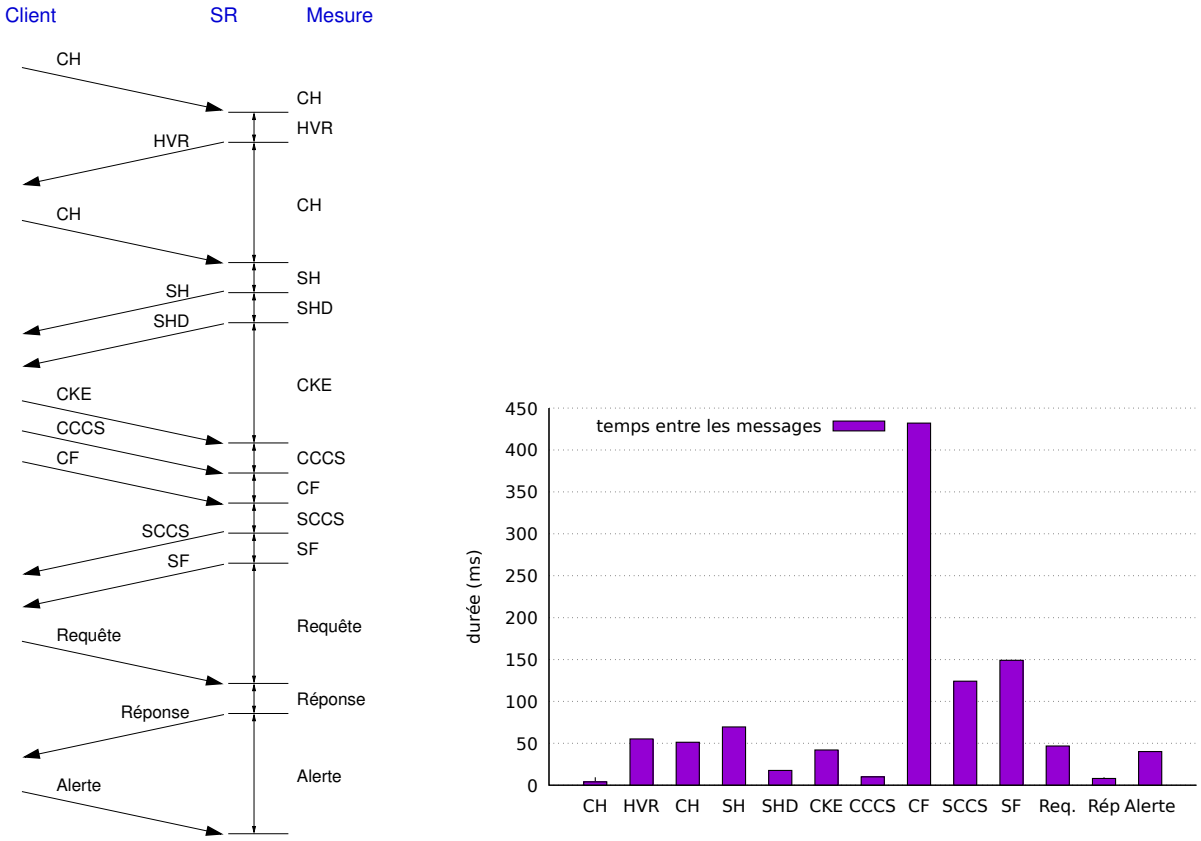
9.6.4 Analyse de la durée d'une requête dans CASAN-S

Au-delà des opérations cryptographiques, le tableau 9.3 montre les durées des différentes opérations lors d'une requête dans l'architecture CASAN-S. Le délai observé avant l'envoi de la réponse correspond à la somme des durées de déchiffrement de la requête (environ 3 ms) puis de chiffrement de la réponse (environ 2,2 ms), ainsi que l'attente moyenne avant l'envoi d'un message sur le réseau (1,12 ms). L'analyse des communications via le logiciel Wireshark montre que la requête est envoyée à 9 ms, ce qui correspond au temps pour terminer la connexion TLS du client avec le maître. La durée totale de communication observée est de 34,1 ms : 21,5 ms sont justifiées par les calculs et les mesures, les 12,6 ms restantes sont en partie liées au démarrage et à l'arrêt du programme qui fait la requête, ainsi qu'au mécanisme utilisé pour gérer les requêtes HTTP.

9.6.5 Opérations longues lors d'une requête dans A-DTLS

Nous identifions les opérations longues dans la requête CoAP protégée par DTLS en analysant le temps d'attente entre les messages. La figure 9.11a illustre les mesures des durées de traitement et d'attente entre deux messages (reçus ou envoyés), et la figure 9.11b montre les résultats. Par exemple, le message DTLS *Client Change Cipher Spec* est traité seulement quelques millisecondes après le message *Client Key Exchange*, ils sont envoyés l'un à la suite de l'autre par le client, et le serveur ne fait aucun calcul important entre les deux. Le temps indiqué est donc uniquement le temps de communication sur la liaison série, et dans une moindre mesure celui de la communication réseau

8. Les durées des opérations cryptographiques ont été mesurées via la commande `openssl speed`.



(A) Mesures effectuées.

(B) Temps observés entre les messages.

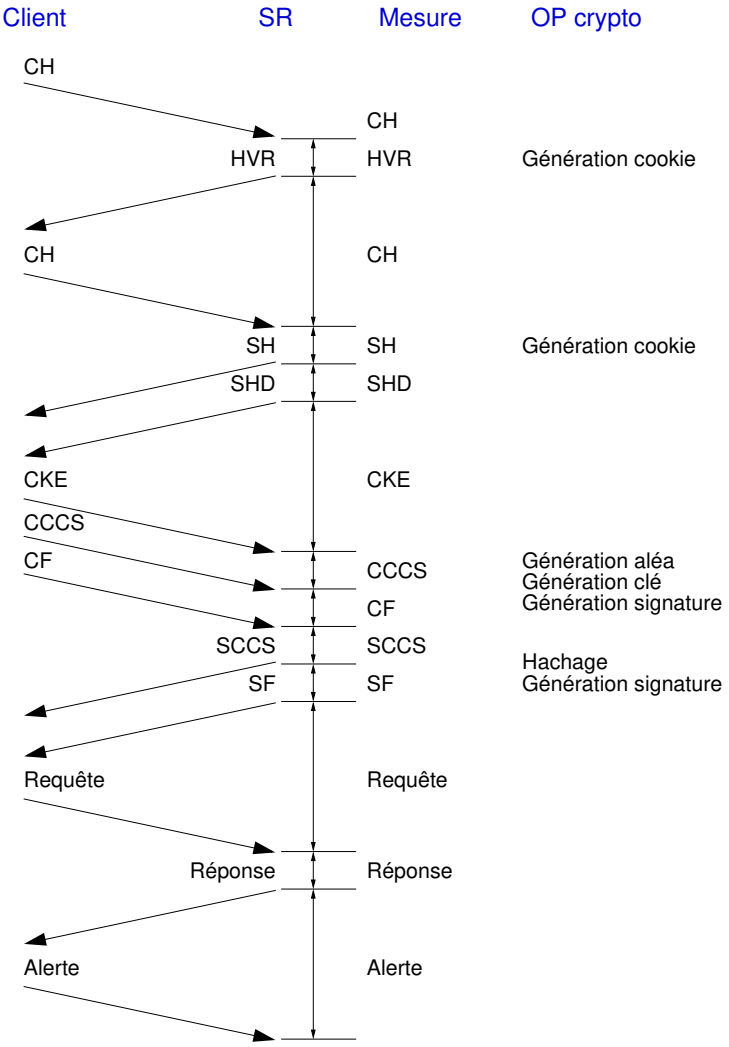
FIGURE 9.11 – Temps entre les messages sur le serveur de ressources dans l'architecture A-DTLS.

(bien plus rapide). Contrairement aux messages évoqués précédemment, la figure montre un temps de plusieurs centaines de millisecondes avant le traitement du message *Client Finished*, ce qui correspond au temps de génération des clés de chiffrement (qui survient sur le serveur après la réception du message *Client Change Cipher Spec*). Cette figure montre distinctement l'importance des durées des opérations cryptographiques, bien plus que l'attente avant envoi et le transfert de messages, comme lors du démarrage du réseau dans l'architecture maître-esclave. Le tableau B.2 (page 125) montre le lien entre les tailles des messages et le temps d'échange sur le réseau 802.15.4 et sur le lien série.

9.6.6 Analyse de la durée d'une requête dans A-DTLS

Pour comprendre la durée de la première requête dans l'architecture A-DTLS, il faut analyser les différentes étapes de la requête. La figure 9.12a (page 109) montre les opérations cryptographiques effectuées lors d'une requête dans l'architecture A-DTLS, et le tableau 9.12b détaille les durées que nous observons et que nous calculons lors d'une requête. La liaison série implique un biais, le temps de transfert sur cette liaison est limité à 12 % du temps total observé pour une requête. La somme des durées des opérations cryptographiques et de la gestion du cookie représentent 821,61 ms, soit 78,47 % du temps total observé. Ce pourcentage monte à 89,5 % si nous retirons le biais engendré par la durée de transfert sur la liaison série. Nous pouvons également voir que la durée minimale calculée d'une requête est proche de la durée observée (1004 ms calculée, 1047 ms observée)[9].

9. Un essai de liaison à 115200 bauds entre le routeur et l'hôte a permis d'améliorer les performances et d'observer un temps de requête à 968 ms en moyenne (avec un écart type de 2 ms), mais le taux de succès des requêtes est passé de 100 % à 70 % dans un environnement de seulement un nœud. Pour cette raison, nous avons limité les expérimentations sur notre plateforme avec une liaison entre le routeur et l'hôte à 57600 bauds. En reprenant les valeurs observées et calculées précédemment, nous arrivons à une



(A) Mesures effectuées.

Opération	Durée (ms)	Obtention du résultat
génération du cookie	53,79 (×2)	mesure (écart type : 0,008 ms)
génération de l'aléatoire	2,75	mesure (écart type : 0,005 ms)
hachage	32,5	mesure (écart type : 0,024 ms)
génération de clés	430	mesure (écart type : 0,004 ms)
génération du message Finished	129	mesure (écart type : 0,002 ms)
vérification du message Finished	107	mesure (écart type : 0,002 ms)
échange sur port série	130	$temps_{\text{transfert}} = \frac{taille_{\text{données}}}{vitesse} = \frac{753}{5760}$
échange sur réseau 802.15.4	38	$temps_{\text{transfert}} = \frac{taille_{\text{données}}}{vitesse} = (1124 + 6 \times 13) \times 0,03125$
attente sur le réseau 802.15.4	14,56	$13 \times 1,12$
durée de chiffrement	4,956	mesure (messages : Finished, Réponse)
durée de déchiffrement	7,824	mesure (messages : Finished, Requête, Alerte)
Total calculé	1004.17	somme des durées précédentes
Total observé	1047	mesure (écart type : 3 ms)

(B) Résultats obtenus.

FIGURE 9.12 – Durées calculées et observées des opérations cryptographiques pour une requête dans l'architecture A-DTLS sur un nœud. Le calcul des durées est effectué en tenant compte d'une liaison série à 57600 bauds entre le routeur et l'hôte. Toutes les mesures sont réalisées sur 100 essais.

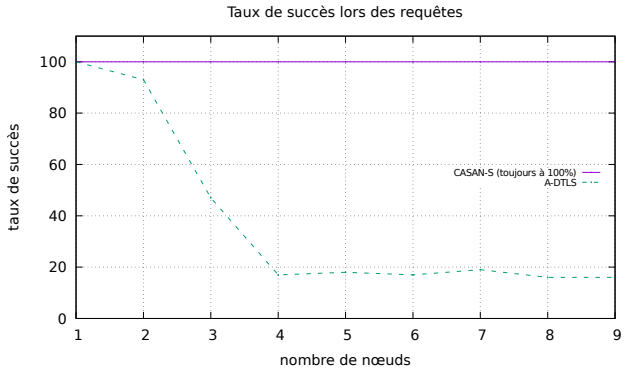


FIGURE 9.13 – Taux de succès des requêtes dans les architectures A-DTLS et CASAN-S.

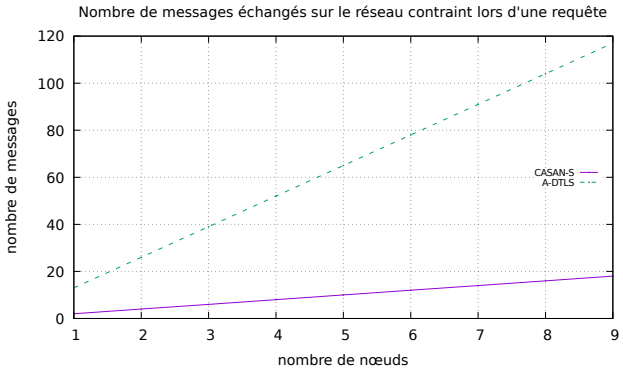


FIGURE 9.14 – Calcul du nombre de messages échangés sur le réseau contraint lors d'une requête.

9.6.7 Comparaison des architectures sur les durées de requête

Contrairement à une architecture de référence, les opérations cryptographiques effectuées lors d'une requête dans une architecture maître-esclave sont minimales pour le matériel contraint. Une requête est réduite aux simples opérations de chiffrement et déchiffrement d'un message sur l'esclave (soit 5 à 6 ms). Nous pouvons conclure que la connexion pré-établie entre le maître et ses esclaves permet un gain considérable par la suite lors des requêtes des clients.

Une interrogation se pose avant le choix d'une architecture : est-ce plus important de démarrer le réseau rapidement (en moins d'une centaine de millisecondes), ou de répondre le plus rapidement possible aux requêtes des clients ?

9.6.8 Taux de complétion des requêtes

Le taux de succès lors de requêtes dans l'architecture A-DTLS chute rapidement avec l'augmentation de la taille du réseau, la figure 9.13 montre l'évolution de ce taux pour les deux architectures. Lors d'une requête, la perte d'un paquet implique une connexion qui restera inachevée et qui sera donc comptabilisée en erreur : les mécanismes de retransmission dans les protocoles CoAP et DTLS n'ont pas un délai suffisamment agressif pour terminer la communication dans le temps imparti de 10 secondes. Les deux architectures sont concernées par ce problème : si un paquet arrive lors d'une communication série, cela provoque une perte du message. Puisque le nombre de messages échangés sur le réseau contraint est beaucoup plus important dans l'architecture A-DTLS que dans CASAN-S (13 messages contre seulement 2), le taux de succès chute considérablement. La figure 9.14 montre la différence de nombres théoriques de messages échangés entre A-DTLS et CASAN-S sur le réseau contraint lors d'une requête d'un client, ce qui explique le plus faible taux de réussite pour A-DTLS.

9.6.9 Conclusion

Dans cette section, nous avons observé que les requêtes dans CASAN-S sont plus de 30 fois plus rapides que dans A-DTLS. La différence des résultats s'explique par le nombre de messages échangés sur le réseau contraint et les opérations cryptographiques nécessaires sur les nœuds contraints : deux métriques en faveur de CASAN-S.

valeur théorique d'échange de 952 ms avec une liaison série de 115200 bauds, et nous voyons que nous conservons un écart de 16 ms avec les observations.

9.7 Synthèse des deux scénarii

Pour procurer une vision synthétique des deux scénarii étudiés dans ce chapitre, nous proposons une vision globale à travers plusieurs métriques pour mieux comprendre l'avantage d'une architecture maître-esclave pour des déploiements avec de nombreux clients. Nous verrons également que l'architecture maître-esclave est intéressante également avec des clients distants.

9.7.1 Charge acceptée par les deux architectures

Cette charge peut se mesurer en nombre de clients se connectant par seconde pour récupérer une donnée, ainsi qu'en nombre de requêtes possibles (une fois la connexion établie). Dans l'architecture A-DTLS, une requête d'un client implique 10 paquets pour la connexion, puis 2 messages de requête et enfin un dernier message pour arrêter la connexion ; ces 13 messages représentent une taille cumulée de 1124 octets. La durée passée sur le réseau lors d'une requête correspond à la somme des durées d'attente (CSMA-CA) et d'émission, comme calculée comme suit :

$$\begin{aligned} T_{\text{usage du réseau}} &= T_{\text{attente}} + T_{\text{émission}} \\ &= \text{nombre}_{\text{messages}} \times T_{\text{attente 1 message}} + \text{nombre}_{\text{symboles}} \times T_{\text{symbole}} \end{aligned} \tag{9.1}$$

Soit dans le cas de l'architecture A-DTLS :

$$\begin{aligned} T_{\text{usage du réseau connexion A-DTLS}} &= 13 \times 1.12 + 1124 \times 0.03125 \\ &= 49,6 \text{ ms} \end{aligned} \tag{9.2}$$

Ce temps est incompressible et représente l'usage du réseau sur un médium partagé lors d'une requête d'un nouveau client. Nous pouvons en déduire que le nombre maximum de clients pouvant se connecter par seconde pour effectuer une requête dans l'architecture est de $\frac{\text{T}}{\text{durée de connexion}}$. Pour l'architecture A-DTLS, ce nombre s'élève à $\frac{1}{0.0496} = 20$ nouveaux clients par seconde au maximum. Le calcul effectué prend seulement en compte la durée des échanges sur le réseau contraint, qui est une limite indépassable et indépendante de la puissance de calcul des nœuds.

Si la connexion est maintenue, et que les clients continuent à envoyer des requêtes, le temps d'usage du réseau est alors de :

$$\begin{aligned} T_{\text{usage du réseau requête A-DTLS}} &= 2 \times 1.12 + 183 \times 0.03125 \\ &= 7,9 \text{ ms} \end{aligned} \tag{9.3}$$

Cette fois, seuls deux messages sont transférés sur le réseau (la requête et la réponse), soit 183 octets. Un temps d'usage du réseau de $7,9$ ms par requête permet jusqu'à 125 requêtes par seconde dans l'architecture A-DTLS.

Dans l'architecture CASAN-S, un client qui se connecte ou se déconnecte n'implique pas de communication sur le réseau contraint. Le nombre maximal de clients se connectant à l'architecture est lié à la capacité de calcul du maître, un nœud non contraint, qui peut paralléliser les connexions en plus d'effectuer les calculs rapidement ; la limite imposée au nombre de clients est la capacité de communication sur le réseau contraint. Le nombre maximal de connexions et le nombre de requêtes par seconde sont donc jugés identiques.

$$\begin{aligned} T_{\text{usage du réseau CASAN-S}} &= 2 \times 1.12 + 82 \times 0.03125 \\ &= 4,8 \text{ ms} \end{aligned} \tag{9.4}$$

Le temps d'usage du réseau pour une requête dans l'architecture CASAN-S est de $4,8$ ms, ce qui permet jusqu'à 208 requêtes par seconde.

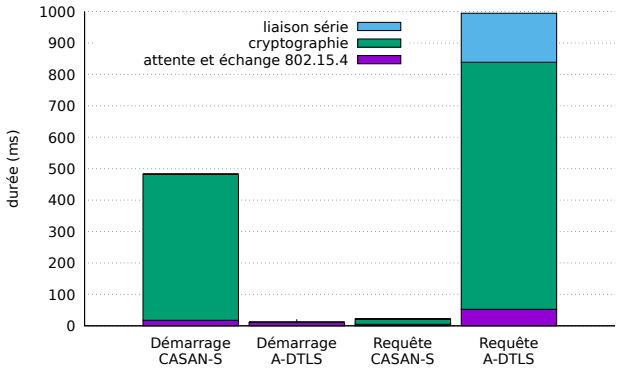


FIGURE 9.15 – Durée comparée des échanges sur le réseau contraint, avec séparation des différentes phases.

L'architecture CASAN-S accepte donc 10 fois plus de connexions suivie d'une requête par seconde sur le réseau, indépendamment des nœuds sollicités. De plus, elle permet jusqu'à 66 % de requêtes supplémentaires par seconde pour des clients déjà connectés par rapport à A-DTLS, sans compter les clients qui voudront des données en cache.

Métrique	**Signification**	**A-DTLS**	**CASAN-S**
$T_{\text{usage du réseau connexion}}$	Connexion d'un client puis requête sur un nœud	49,6	4,8
$T_{\text{usage du réseau requête}}$	Requête d'un client déjà connecté	7,9	4,8

TABLE 9.4 – Résumé des principaux indicateurs évalués.

La figure 9.4 résume les indicateurs de charge d'un réseau.

9.7.2 Où passe le temps ?

Pour résumer l'importance de l'architecture maître-esclave pour le temps de requête, nous récapitulons sur la figure 9.15 les durées comparées pour les deux architectures des échanges lors du démarrage du réseau et lors de la première requête d'un nœud. La figure met en perspective les durées de démarrage ou de requête sur le réseau contraint, en différenciant les durées de calcul, d'échange (et d'attente), ainsi que la durée de transfert sur la liaison série. Cette figure met également en évidence l'importance de réduire les opérations cryptographiques, et dans une moindre mesure de limiter le nombre d'échanges sur le réseau contraint.

Au-delà de la première requête dans une architecture de référence, si le client maintient la connexion, alors les requêtes suivantes s'effectuent dans un temps proche de celui d'une architecture maître-esclave. En ne prenant en compte que les opérations de chiffrement et déchiffrement, l'attente et le transfert de la requête sur le réseau contraint, une requête dans l'architecture de référence A-DTLS prend 14,7 ms, contre 11,6 ms dans une architecture maître-esclave CASAN-S [10]. L'architecture CASAN-S est avantagée lors de requêtes de clients déjà connectés grâce à ses messages dépourvus d'en-têtes de couches réseau et transport, et avec un en-tête de couche applicative intégrant le mécanisme de sécurité ; ces messages sont alors plus courts et prennent moins de temps lors du transfert.

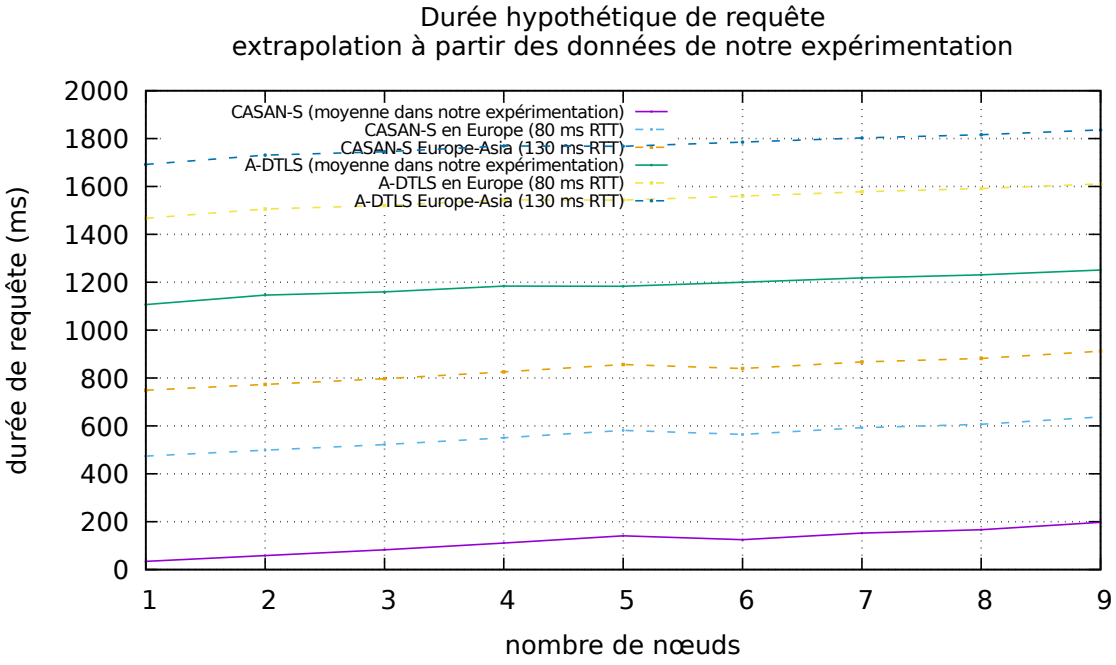


FIGURE 9.16 – Durées hypothétiques de requête avec un client non local. Deux distances sont considérées, à savoir deux points sur Internet en Europe (environ 80 ms de délai), et entre l'Europe et l'Asie (environ 130 ms de délai).

9.7.3 Comportement des architectures avec un client distant

Enfin, pour compléter cette analyse, nous décrivons le comportement des architectures avec un client distant. Jusqu'à présent, nous avons considéré le client comme local au réseau, ou ne faisant pas partie du réseau contraint mais sans délai avec le routeur. Si un client sur Internet envoie des requêtes à nos architectures, cela ajoute des délais supplémentaires. La figure 9.16 montre les temps de requête avec des délais supplémentaires. Le choix des durées ajoutées est lié aux temps d'aller-retour de messages sur Internet entre deux pays en Europe (RTT de 80 ms), et entre Europe et Asie (RTT de 130 ms). Nous pouvons voir qu'en ajoutant un délai entre Europe et Asie, CASAN-S obtient toujours de meilleurs délais qu'une requête en local avec l'architecture A-DTLS. La distance du client par rapport au réseau contraint implique des délais moins importants dans A-DTLS : CoAP et DTLS nécessitent 5 RTT contre 6 pour HTTPS dans l'architecture CASAN-S. Cependant, l'architecture CASAN-S comprend plusieurs piles de protocoles et réalise une traduction : nous pouvons remplacer HTTPS par CoAPS, comme utilisé actuellement dans A-DTLS pour comparer ces deux architectures sur un pied d'égalité. Dans ce cas, le nombre de RTT est le même dans les deux architectures, l'évolution du temps de requête est donc proportionnel à l'augmentation des échanges sur le réseau contraint, qui est bien plus important dans les architectures de référence.

9.7.4 Conclusion

Dans toutes les situations observées, CASAN-S a l'avantage lors de la connexion de clients et lors de requêtes. Dans cette section, nous avons calculé que l'architecture CASAN-S accepte bien plus de nouveaux clients à la seconde que A-DTLS grâce au faible nombre de messages sur le réseau contraint

10. Le calcul du temps de réponse à une requête dans l'architecture A-DTLS est de : 3.82 ms de déchiffrement, 2.99 ms de chiffrement, 2.24 ms d'attente avant envoi sur IEEE 802.15.4, et 5,71 ms de durée d'échange de 183 octets sur le réseau contraint (183×0.03125). Le calcul est similaire pour l'architecture CASAN-S : 3.82 ms de déchiffrement, 2.99 ms de chiffrement, 2.24 ms d'attente avant envoi, et 2,56 ms de durée d'échange de 82 octets sur le réseau contraint (82×0.03125).

et à l'authentification sur un nœud non contraint. De plus, CASAN-S accepte également un plus grand nombre de requêtes à la seconde venant de clients déjà connectés que l'architecture A-DTLS, car les messages ont des en-têtes plus courts et donc ont un plus petit temps de transfert. Enfin, la présence de clients distants est également à l'avantage de CASAN-S.

9.8 Conclusion

La première contribution de ce chapitre est la comparaison des architectures CASAN, CASAN-S et A-DTLS (architecture basée sur DTLS, vue au chapitre 4) d'un point de vue qualitatif. Notre nouvelle architecture présente tous les avantages d'une architecture de référence telle que A-DTLS auquel nous aurions ajouté un cache de données, et plus encore : CASAN-S nécessite moins de prérequis (et donc un code plus simple), moins de messages sur le nœud contraint lors d'une requête (seulement 2 messages contre 12 à la première connexion d'un client), et une charge utile plus importante (de 18,1 à 33 %). Les architectures CASAN et CASAN-S n'obligent pas le client à connaître l'identité de l'esclave pour effectuer une requête. Ces architectures ne nécessitent qu'une seule connexion du client au maître, quelles que soient les requêtes que le client souhaite effectuer. Elles ont également un mécanisme intégré de contrôle d'accès, et ne nécessitent pas d'autoconfiguration d'adresses, ni de DHCP, ni de DNS. Enfin, le principal avantage de CASAN-S par rapport à CASAN réside dans la réduction des en-têtes, permettant un gain de charge utile.

La seconde contribution a été de mesurer les durées de démarrage du réseau et de requête des architectures CASAN-S et A-DTLS. Les mesures montrent un net avantage en faveur de l'architecture maître-esclave, qui est plus efficace lors des requêtes et rend presque insignifiant son délai de démarrage.

L'architecture A-DTLS a l'avantage lors du démarrage d'un nœud dans le réseau (17 ms pour A-DTLS, 490 ms pour CASAN-S), qui ne s'opère qu'une fois entre deux redémarrages des nœuds. Aucune opération cryptographique n'est nécessaire, les nœuds demandent le préfixe du réseau pour créer leur adresse, puis la donnent au routeur 6LoWPAN pour l'ajouter à la table de routage. À l'inverse, l'architecture CASAN-S nécessite d'établir une connexion sécurisée entre le maître et l'esclave, puis d'annoncer de manière sécurisée les ressources disponibles sur le nœud. La durée des opérations cryptographiques représente la majeure partie de la durée totale de l'ajout d'un nœud dans CASAN-S, sans quoi les durées seraient équivalentes dans les deux architectures. Cette connexion sécurisée est la seule établie dans une architecture maître-esclave, par la suite le nœud contraint n'a plus à effectuer d'opérations cryptographiques longues.

La durée d'une connexion suivie d'une requête est plus de 30 fois plus courte dans l'architecture CASAN-S (34 ms pour CASAN-S, 1047 ms pour A-DTLS) ; l'inconvénient du démarrage lent est donc amorti dès la première connexion du premier client. CASAN-S a l'avantage lors d'une connexion d'un client et lors d'une requête car seule la requête et sa réponse sont échangées sur le réseau contraint au travers d'une communication sécurisée déjà établie avec l'esclave, qui n'effectue alors que des opérations cryptographiques légères (chiffrement et déchiffrement symétriques). Contrairement à CASAN-S, l'architecture A-DTLS nécessite l'authentification des clients sur les serveurs de ressources et la création d'un secret partagé ; les opérations cryptographiques composent alors 89,5 % de la durée d'une connexion suivie d'une requête.

Les différences de performances entre les deux architectures s'expliquent avant tout par les opérations cryptographiques nécessaires sur les nœuds contraints. Ces opérations ralentissent le temps de stabilisation d'une architecture maître-esclave, mais cette architecture est favorisée lors d'une requête : la conception maître-esclave permet une communication simplifiée dans le réseau contraint (les messages n'ont pas de couche réseau ni transport, et sont moins nombreux car sur une connexion sécurisée déjà établie), et l'architecture nécessite peu de cryptographie sur les nœuds contraints.

D'après nos travaux, une architecture de référence est alors pertinente seulement dans les cas suivants :

— le déploiement ne permet pas l'introduction d'un maître (trop coûteux, ou impossibilité technique) ;
— la sécurité de bout-en-bout (du client au serveur de ressources) est nécessaire ;
— le temps de stabilisation du réseau est critique, et doit s'effectuer le plus rapidement possible.

Pour tous les autres cas étudiés, une architecture maître-esclave est à préférer : elle permet la connexion puis la requête de clients dix fois plus rapidement et une fréquence de requêtes jusqu'à 66 % plus rapide lorsque les client restent connectés. Ces résultats sont très en faveur de l'architecture maître-esclave malgré l'utilisation d'algorithmes asymétriques pour faciliter la gestion de l'authentification et du partage de clés, ce qui s'explique par l'utilisation d'un nœud non contraint comme maître. De plus, cette architecture intègre un cache de données, et permet de ne maintenir qu'une seule connexion pour récupérer des informations sur plusieurs capteurs.

Chapitre 10

Conclusion et perspectives

Le principe de l'Internet des Objets est de pouvoir accéder à des données, ou envoyer des commandes, à des objets depuis Internet. Nous pouvons par exemple retrouver ces objets au sein d'entreprises nécessitant des mesures régulières (par exemple la température d'une pièce) ou un asservissement de machines à distance (par exemple pour un contrôle centralisé de la ventilation). Les données récoltées sur les capteurs, ou envoyées aux actionneurs, ne doivent pas être divulguées ; les clients et les objets doivent être authentifiés, leurs actions autorisées, et toute communication doit être protégée. La difficulté de cette tâche dans l'IoT vient des contraintes sur les objets, qui ont de fortes limitations de mémoire, calcul et communication.

Deux conceptions d'architecture sont possibles : dans une architecture de référence, l'objet est autonome et gère l'authentification, l'autorisation et la communication sécurisée avec les clients, en plus de leur fonction de capteur ou d'actionneur. Dans une architecture maître-esclave, l'objet est dédié à sa fonction de capteur ou d'actionneur, et délègue le reste à un nœud non contraint.

Dans cette étude, nous nous sommes focalisés sur un déploiement de technologies de communication et de sécurité des échanges standards, décrites aux chapitres 2 (page 5) et 3 (page 17). En effet, ces technologies représentent des briques de base de nombreux déploiements et apportent des contraintes de débit, de fréquence des messages, de taille des messages échangés ou encore des contraintes de mémoire par exemple. Par la suite, ces contraintes forment nos critères de comparaison des architectures existantes. En l'absence d'une définition de ce qu'est une architecture de sécurité, notre première contribution [100] a été d'en proposer une (chapitre 4 page 29). Cette définition a introduit la terminologie employée par la suite lors de l'étude et la comparaison des différentes solutions de sécurité au chapitre 5 (page 41). La comparaison a permis de créer un guide au chapitre 6 (page 65) qui classe les architectures en fonction des contraintes de déploiement (faible mémoire, faible MTU, nombre de clients, etc.).

Nous avons également contribué à l'amélioration d'une architecture de sécurité en optimisant son mécanisme de sécurité [95], et nous avons obtenu un gain de 19,9 % de temps de connexion et 8,2 % de charge utile supplémentaires pour les données. Cependant cette amélioration est loin d'égaler les avantages qu'offre une architecture maître-esclave, ce que nous montrons dans une contribution en cours de rédaction et dont nous venons de présenter les résultats.

1. Avantages d'une architecture maître-esclave

Nous avons comparé une architecture de référence et une architecture maître-esclave et les avantages d'une architecture maître-esclave sont multiples. Utiliser un maître non contraint pour l'authentification, l'autorisation et cache de données permet en premier lieu de **simplifier les couches de communication** dans les environnements contraints. En effet, un maître pouvant assurer la communication avec ses esclaves (les nœuds contraints) peut s'abstraire de la contrainte de communication de bout-en-bout nécessitant la même pile de protocoles du client au serveur de ressources.

Ne pas entretenir une communication de bout-en-bout permet au client de ne devoir **faire confiance qu'au maître**, qui se situe dans le même réseau de confiance (le même domaine) que le serveur de ressources. La responsabilité revient ensuite au maître de vérifier la confiance accordée au reste de l'architecture. La sécurité est ainsi plus aisée grâce à une **gestion centralisée** des clients (authentification, autorisation) et du matériel cryptographique des serveurs de ressources ; tout changement de secret partagé s'effectue entre le maître et un client ou un serveur de ressources.

Ensuite, la **gestion des clés est simplifiée** car les serveurs de ressources n'ont pas à connaître les clients, c'est-à-dire qu'il n'y a pas de clé de chiffrement à maintenir par client, et inversement, les clients n'ont à retenir aucune information à propos des serveurs de ressources. De plus, l'authentification du domaine serveur auprès du client peut être réalisée avec des algorithmes asymétriques, par exemple à l'aide de certificats avec une longue période de validité, permettant de réduire la maintenance du matériel cryptographique dans l'architecture. En effet, **tous les algorithmes sont disponibles** pour sécuriser la connexion entre le client et le domaine contraint car le domaine serveur peut traiter les clients sans limite de capacité de calcul.

Lors d'une connexion, le client échange des messages uniquement avec le maître et la connexion entre le maître et l'esclave est déjà établie. De ce fait, **aucun calcul cryptographique lent** n'est nécessaire sur les nœuds contraints et **aucun message de connexion ne doit être envoyé sur le réseau contraint**. À l'inverse, une architecture de référence nécessite la création d'une clé de chiffrement, la génération et la vérification d'une signature sur les nœuds contraints à chaque connexion d'un client. Par conséquent, comme nous avons pu l'observer au chapitre 9 (page 93), **une connexion suivie d'une requête est plus de 30 fois plus rapide** sur un réseau maître-esclave que dans une architecture de référence. De plus, lorsque les clients restent connectés aux serveurs de ressources, l'architecture maître-esclave permet d'être jusqu'à **66 % plus rapide lors de requêtes** et offre une **charge utile plus conséquente** dans les messages (de 18 à 33 % de charge utile supplémentaire dans notre comparaison entre A-DTLS et CASAN-S).

2. L'IoT sans les problèmes

Une architecture de référence nécessite une pile de communication compatible entre les clients et les serveurs de ressources. Dans une architecture maître-esclave, l'esclave n'a pas de contrainte d'accessibilité IP car le client ne le contacte pas, ni de contrainte de couche transport : le maître et l'esclave s'accordent sur la manière de communiquer toutes les informations, et tout type de donnée est échangé d'une seule manière, qui est CoAP dans l'architecture CASAN(-S). La communication entre le maître et l'esclave nécessite seulement que les deux soient identifiés et puissent échanger des messages via une technologie de niveau physique et liaison telle que vue au chapitre 2, de manière sécurisée.

Une architecture de référence réutilise des mécanismes existants pour communiquer sur Internet avec le serveur de ressources, comme DTLS par exemple. Ces mécanismes peuvent être réutilisés dans une communication entre le maître et ses esclaves, mais ils apportent des fonctionnalités déjà présentes dans les protocoles utilisés (comme la fragmentation, présente dans DTLS et CoAP) ou superflues dans une architecture maître-esclave (comme le cache de données, qui est déjà effectué par le maître). Nous avons montré qu'il est possible de réduire à une seule couche de communication les échanges entre le maître et ses esclaves avec CASAN-S, sans fonctionnalité superflue, réduisant ainsi les contraintes sur la mémoire des esclaves et sur le réseau.

Le maître seul est exposé sur Internet, réduisant considérablement la surface d'attaque depuis l'extérieur du réseau : il s'occupe de filtrer les requêtes et de contrer un déni de service si le domaine est attaqué. Les esclaves sont ainsi protégés des requêtes malveillantes, puisqu'elles sont en premier lieu interprétées, traduites et éventuellement filtrées par le maître.

3. Inconvénients d'une architecture maître-esclave

Une architecture maître-esclave souffre de quelques inconvénients, comme la nécessité de déployer un nœud non contraint pour le maître. De plus, une telle architecture ne permet pas la sécurité de bout-en-bout, cependant cela n'est pas nécessaire car le maître fait partie du domaine serveur, ce n'est pas un intermédiaire technique contrôlé par une autre autorité, par conséquent le client lui fait aussi confiance qu'à un objet. Enfin, le démarrage du réseau est ralenti par la mise en place d'une

communication sécurisée entre le maître et l'esclave, comme non l'avons mesuré au chapitre 9 pour l'architecture CASAN-S ; l'impact de cette connexion peut être amorti en conservant des sessions sur le long terme, l'esclave n'a pas à établir une nouvelle session à chaque démarrage.

4. Perspectives

Notre travail pourrait être étendu pour fournir automatiquement des propriétés sur les architectures de sécurité en rendant la description (les relations) exécutable par une machine de Turing. Les propriétés étudiées seraient par exemple le nombre de messages échangés, la durée des échanges ou encore la taille des informations conservées sur chaque matériel. Pour arriver à une description exécutable, nous pourrions dans un premier temps classer les informations, notamment celles liées aux sessions (algorithmes, clés, identifiants, etc.) pour abstraire les détails des protocoles et simplifier les raisonnements sur les échanges. Nous pourrions également classer les messages par médium utilisé pour y attribuer un coût : une liaison Ethernet est plus rapide et les erreurs sont moins fréquentes que sur IEEE 802.15.4, et le nombre de sauts devrait également être pris en compte. Ensuite, nous pourrions déterminer sur quel matériel se situe une information en regroupant les entités qui les composent, nous pourrions donc savoir quel matériel retient les informations des clients et les authentifie. De plus, cela permettrait de tracer les informations des sessions et le temps pendant lequel elles sont retenues et renouvelées. Avec le classement des informations, la connaissance des médiums utilisés pour les échanges et le traçage des informations sur les matériels, nous pouvons modéliser une architecture de sécurité. Le niveau de précision employé pour la modélisation doit correspondre aux propriétés visées par cette modélisation. Par exemple, si nous recherchons une durée d'échange alors les tailles des messages devront être conservées, mais si nous recherchons le nombre d'échanges alors les tailles ne sont pas nécessaires. Ce premier approfondissement de notre définition permettrait d'ouvrir de nouvelles voies à nos travaux.

Notre définition et notre manière de décrire les architectures de sécurité peuvent également être vues à travers le prisme de la mise à l'échelle. Une mise à l'échelle correspond à la capacité de gérer un grand nombre d'objets (serveurs de ressources, clients, fréquence de requêtes) sans dégradation significative des performances. Les méthodes pour arriver à une bonne mise à l'échelle sont par exemple de diminuer le nombre et le volume des échanges sur un réseau contraint, ne pas limiter le domaine de requêtes à un seul réseau, ou encore répliquer les matériels et répartir la charge. Cette répartition de la charge peut être effectuée par type ou par sujet de données, par provenance ou destination, ou hiérarchiquement avec une cascade de serveurs. Un mécanisme qui passe à l'échelle est par exemple QEST [101] qui permet une réplication des nœuds qui répondent aux requêtes des clients car ils sont sans état, ou bien le chaînage de maîtres dans l'architecture CASAN qui permet d'étendre le domaine de requêtes à plusieurs réseaux (répartition hiérarchique).

À quel point une architecture accepte une charge de clients, ou de serveurs de ressources ? Pour répondre à ces questions, nous pouvons attribuer un score de mise à l'échelle aux architectures, par exemple en étudiant la redondance possible des nœuds, en déterminant quelles sont les données à synchroniser pour répartir la charge, en associant un coût aux médiums utilisés et un coût à la synchronisation des informations sur les matériels qui doivent être dupliqués pour accepter plus de clients, etc. Ce score pourrait être calculé automatiquement grâce au travail sur une description exécutable : en connaissant les informations échangées, en les ayant classées pour connaître leur sémantique (pour savoir si une information est liée à la session avec un client par exemple), en connaissant également les médiums utilisés et les matériels, nous pouvons déterminer un coût global de mise à l'échelle de l'architecture. Ce travail nous permettrait de répondre à des questions comme : quelle est l'architecture qui accepte le plus de clients ? Celle qui impose le moins de requêtes aux serveurs de ressources ? La réplication de matériel est-elle possible facilement ou est-ce que la synchronisation des données entre les serveurs répliqués est trop importante (volumes de données échangées, fréquence, etc.) ? Ou encore, pour une architecture donnée, quels sont les points les plus importants à améliorer pour passer à l'échelle ?

Annexe A

Modes de chiffrement

Les algorithmes de chiffrement fonctionnent soit en continu, c'est-à-dire qu'ils prennent chaque octet du texte en clair au fil de l'eau et le chiffrent, soit ils fonctionnent par bloc, c'est-à-dire qu'ils prennent en paramètre un certain nombre de bits du message en clair pour les chiffrer. Il y a plusieurs manières de chiffrer les longs messages (dont la taille est supérieure à un bloc).

A.0.0.1 Mode Electronic Codeblock (ECB)

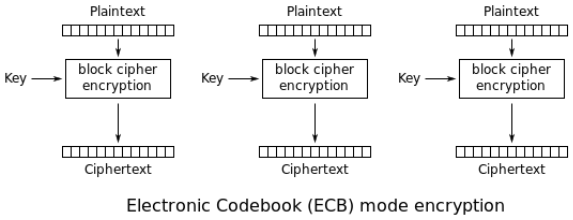


FIGURE A.1 – Chiffrement ECB, extrait de Wikipédia.

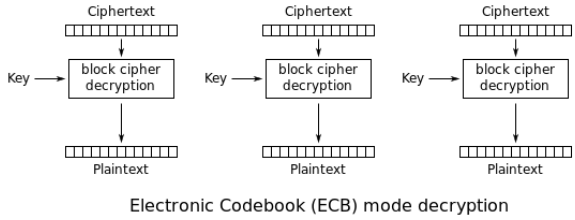


FIGURE A.2 – Déchiffrement ECB, extrait de Wikipédia.

Le mode Electronic Codeblock (ECB) est le mode le plus simple. Chaque partie – bloc – du texte clair est chiffrée avec l'algorithme de chiffrement symétrique choisi (par exemple AES) et le déchiffrement est identique, en remplaçant le texte en clair par le texte chiffré. Les figures A.1 et A.2 montrent le fonctionnement.

Le principal désavantage de cette méthode est que deux messages en clair identiques donneront toujours le même texte chiffré. En conséquence, les motifs ne sont pas cachés et peuvent être utilisés pour comprendre le contenu du texte clair dans son intégralité.

De plus, le mode ECB ne permet pas non plus de protéger contre les attaques par rejeu, puisque deux messages identiques donnent le même message chiffré.

A.0.0.2 Mode Cipher Block Chaining (CBC)

Le chiffrement par chaînage de blocs (Cipher Block Chaining, CBC) utilise un vecteur d'initialisation (IV) constitué d'une suite de bits arbitraires. Ce vecteur est utilisé dans une opération XOR avec le premier bloc du message à chiffrer, puis le bloc est chiffré. Ce bloc chiffré est ensuite réutilisé comme faisant partie du texte chiffré final mais aussi comme vecteur d'initialisation pour le bloc suivant et ainsi de suite jusqu'à arriver à un message entièrement chiffré. Le déchiffrement est effectué en prenant un bloc du texte chiffré et en le faisant déchiffrer par l'algorithme de chiffrement précédemment utilisé,

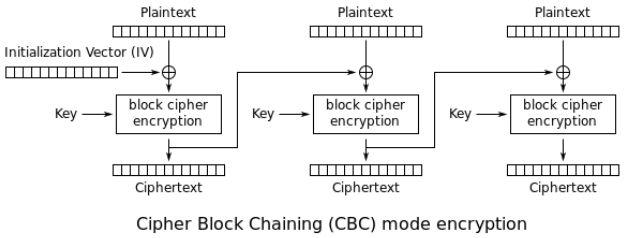


FIGURE A.3 – Chiffrement CBC, extrait de Wikipédia.

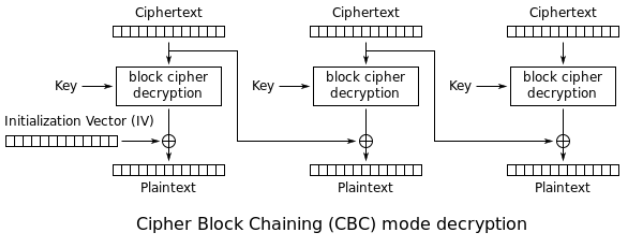


FIGURE A.4 – Déchiffrement CBC, extrait de Wikipédia.

puis une opération XOR est appliquée avec le résultat et le vecteur d'initialisation utilisé précédemment pour le chiffrement. Ce même bloc de texte chiffré est réutilisé comme vecteur d'initialisation pour le bloc suivant. Ce fonctionnement est illustré sur les figures A.3 et A.4.

A.0.0.3 Mode Counter (CTR ou CM)

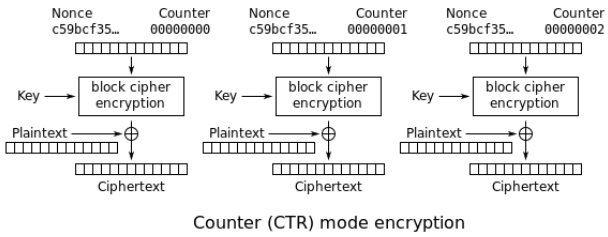


FIGURE A.5 – Chiffrement CTR, extrait de Wikipédia.

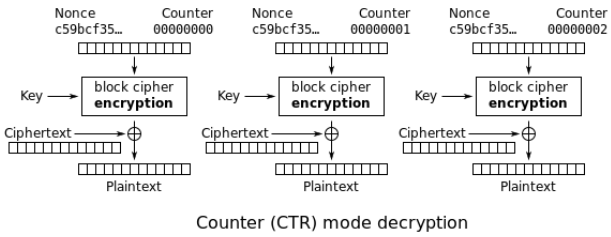


FIGURE A.6 – Déchiffrement CTR, extrait de Wikipédia.

Ce mode de chiffrement implique une *nonce* (une série d'octets unique, comme un vecteur d'initialisation) ainsi qu'un compteur de blocs (un simple nombre s'incrémentant). La nonce est concaténée au compteur et ceci est chiffré (algorithme symétrique) avec la clé de chiffrement, puis on applique une opération XOR avec le résultat et un bloc du message en clair. Le déchiffrement est identique, en remplaçant le texte en clair par le texte chiffré. Il est à noter que la même fonction de chiffrement sert également au déchiffrement du message. Ce fonctionnement est illustré sur les figures A.5 et A.6.

La partie 3.1.2 montre différentes méthodes pour chiffrer un texte long, toujours en prenant des parties du message et en leur appliquant un certain traitement. Ce sont des techniques simples qui peuvent être combinées pour créer d'autres méthodes de chiffrement comme c'est le cas avec des algorithmes de chiffrement authentifié.

Annexe B

Contenu et taille des messages

Pour faire des calculs de performances théoriques pertinents, il faut savoir quelles sont les tailles des messages échangés, ainsi que le médium sur lequel ces messages sont échangés. Cette sous-section se focalise sur le contenu des messages des deux architectures afin de calculer des temps théoriques d'échange.

B.1 Tailles des entêtes de niveau 2

Selon le standard IEEE 802.15.4 (voir section 2.2.3 pour le détail), les adresses MAC peuvent être de 2 ou de 8 octets. Seul ce paramètre varie dans les entêtes de couche 1 et 2 en fonction de l'architecture étudiée dans notre environnement de test. En effet, dans notre environnement de test l'architecture CoAPS utilise des adresses de 8 octets pour créer les adresses IP avec 6LoWPAN, CASAN-S utilise des adresses de 2 octets. Il est possible d'utiliser des adresses de 2 octets avec 6LoWPAN [16], ce qui ferait un gain de 14 octets par message par rapport à notre environnement, soit un gain de temps de transfert de $\frac{14}{31,25} = 0,448$ ms par message. Les messages ont donc une taille d'entête de niveau 2 de :

Taille L2 $= SHR + PHR + MHR + MFR$

Où :

— SHR = préambule (4 octets) et SFD (1 octet)
— PHR = taille de la frame (7 bits) et 1 bit réservé
— MHR = FrameControl (2 octets), numéro de séquence (1 octet), adresse PAN destination (2 octets), IP source et destination (l'adresse PAN source et l'entête de sécurité ne sont jamais précisés dans nos environnements)
— MFR = somme de contrôle (2 octets)

$$\text{Taille L2} = \begin{cases} 17 & \text{si CASAN-S (IP source et destination à 2 octets)} \\ 23 & \text{si CoAPS, broadcast (IP source à 8 octets, 2 pour la destination)} \\ 29 & \text{si CoAPS, unicast (IP source et destination à 8 octets)} \end{cases}$$

B.2 Tailles des entêtes réseau et transport

La figure B.1 montre les entêtes 6LoWPAN des messages CoAPS. Les ports UDP utilisés sont inclus entre 61616 et 61631 afin de pouvoir contenir le numéro UDP en seulement 4 bits.

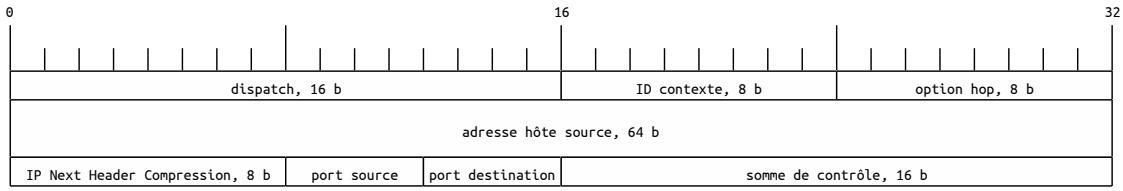


FIGURE B.1 – Format des échanges de l'architecture CoAPS, couche réseau (6LoWPAN) et transport (UDP). L'adresse source est remplacée par l'adresse de destination lors d'un échange venant d'un serveur de ressources vers un client.

	Message	Taille	Taille sur 802.15.4	Taille sur liaison série
CASAN-S	Discover	44	61	52
	Discover Verify	41	58	49
	Discover + Cookie	76	93	84
	Master Assoc	75	92	83
	Slave Assoc	56	73	64
CoAPS	RPL DIS	18	41	-
	RPL DIO	80	103	-
	RPL DAO	53	82	-
	RPL DAO ack	11	40	-

TABLE B.1 – Résumé des tailles des messages CASAN-S et CoAPS, selon le médium sur lequel le message est échangé.

B.3 Tailles des messages lors du démarrage du réseau

Le tableau B.1 référence les tailles des messages CASAN-S en fonction du médium sur lequel ces messages sont échangés. En effet, une fois l'échange effectué sur le réseau 802.15.4, seule une partie du message est ensuite échangée sur le lien série. Les messages échangés sur la liaison série sont liés aux spécificités du périphérique IEEE 802.15.4 : 8 octets sont ajoutés aux tailles des messages reçus et envoyés (adresses sur 16 bits) [1].

B.4 Contenu des messages applicatifs dans CASAN-S

Les figures B.2, B.3, B.4 et B.5 montrent les messages de l'architecture CASAN-S. En vert les entêtes CoAP, en bleu les options non chiffrées, en rouge les options de sécurité (comportant les informations cryptographiques de la session), en blanc les informations chiffrées.

B.5 Tailles des messages lors d'une requête dans A-DTLS

1. Voir la documentation officielle du périphérique [102] section "API Type", page 62 et 63.

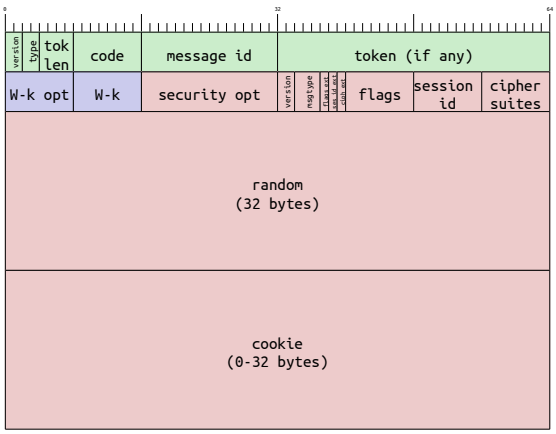


FIGURE B.2 – Message Discover

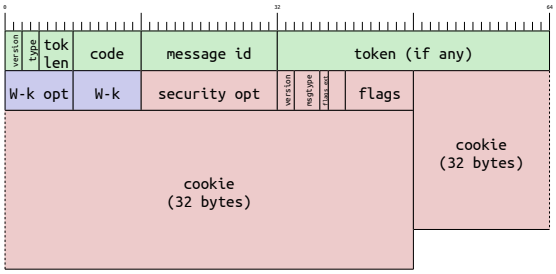


FIGURE B.3 – Message Discover Verify

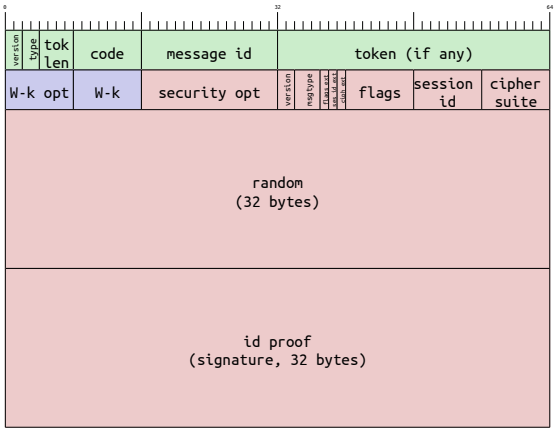


FIGURE B.4 – Message Master Assoc

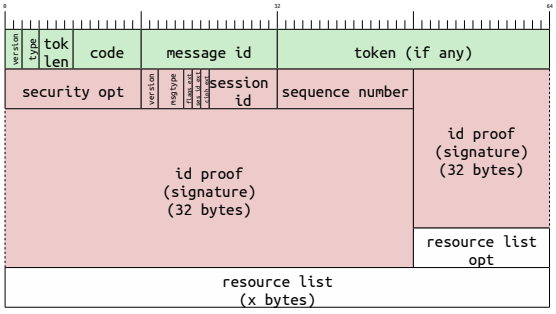


FIGURE B.5 – Message Slave Assoc

message	CH	HVR	CH+C	SH	SHD	CKE	CCCS	CFin	SCCS	SFin	Req	Rép	Alerte	Total
802.15.4	106	90	122	109	71	81	53	92	60	99	92	79	70	1124
série	83	61	93	80	42	52	24	63	31	70	63	50	41	753
délai	-	55,25	51,21	69,49	17,66	42,05	9,93	432,1	124,1	149,1	46,81	8,05	40,16	1045
écart type	-	0,457	1,017	0,627	0,831	2,463	0,607	0,502	0,604	1,021	2,584	0,557	0,95	12,22

TABLE B.2 – Taille d'un message lors de son transfert sur le réseau 802.15.4 et sur le port série. « Délai » correspond à la durée avant traitement d'un message sur le serveur de ressources dans l'architecture CoAPS (moyenne sur 100 essais). La durée indiquée pour le premier Client Hello correspond au temps entre deux tentatives, et n'est pas compté pour la durée totale de la requête. L'écart type est celui des délais de communication.

Annexe C

Influence de la liaison série

Le dispositif expérimental du chapitre 9 (page 93) comporte un facteur limitant : la liaison série entre le module de communication 802.15.4 et la passerelle ou le maître. Dans l'architecture A-DTLS, le démarrage du réseau se passe uniquement dans le réseau contraint (entre le serveur de ressources et le routeur), ainsi la liaison série entre le routeur et l'hôte Linux n'est pas utilisée. Dans l'architecture CASAN-S, le démarrage du réseau implique une communication entre le serveur de ressources et le maître situé sur un hôte Linux. Ce problème est d'autant plus important que le module radio IEEE 802.15.4 n'a qu'une mémoire tampon suffisante pour un nombre limité de paquets, et des paquets sont perdus dès que la vitesse de la liaison série ne permet plus de traiter les paquets aussi vite qu'ils arrivent. L'annexe C détaille l'influence de cette liaison série sur les résultats obtenus.

Pour s'abstraire de ces contraintes, nous avons décidé d'utiliser un Zigduino avec un module Ethernet en remplacement du périphérique IEEE 802.15.4, dans l'architecture CASAN-S [1].

Pour mieux comprendre et quantifier ce phénomène nous avons décidé de mesurer le temps de démarrage des nœuds dans l'architecture CASAN-S sur un réseau de test en variant la vitesse de communication du lien série. Puis nous avons calculé théoriquement le temps de démarrage du réseau en prenant en compte les temps de transfert sur la liaison série et de transfert et d'attente sur le réseau 802.15.4, pour comparer avec les résultats empiriques.

C.1 Résultats

L'histogramme C.1 compare les vitesses théoriques calculées de communication des messages CASAN-S sur le port série et les mesures effectuées en pratique. Les vitesses d'envoi de messages du maître à l'esclave sur le port série ne sont pas présentes sur l'histogramme. Nous observons une augmentation de quelques % du temps théorique dans la pratique, ce qui est lié à la précision de la mesure.

Vitesse en bauds	Temps moyen choisi	Intervalle de temps de retransmission
9600	450	[225, 675]
19200	300	[150, 450]
38400	200	[100, 300]
57600	100	[75, 225]

TABLE C.1 – Intervalles de retransmissions de CASAN-S, correspondant à $[\frac{\text{temps moyen choisi}}{2}, \frac{\text{temps moyen choisi}*3}{2}]$

1. La durée de communication sur le port série est quantifiée et n'est pas le point le plus chronophage lors d'une requête dans A-DTLS.

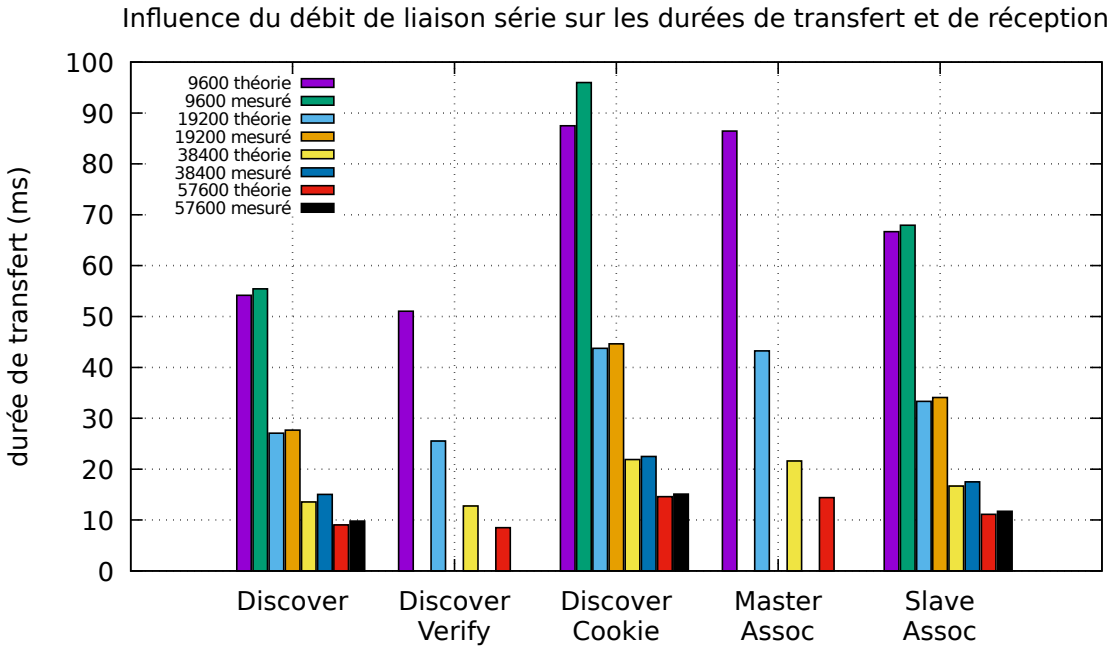


FIGURE C.1 – Durées théoriques et mesurées de communication sur le port série des messages CASAN-S selon la vitesse de communication, sur 100 essais.

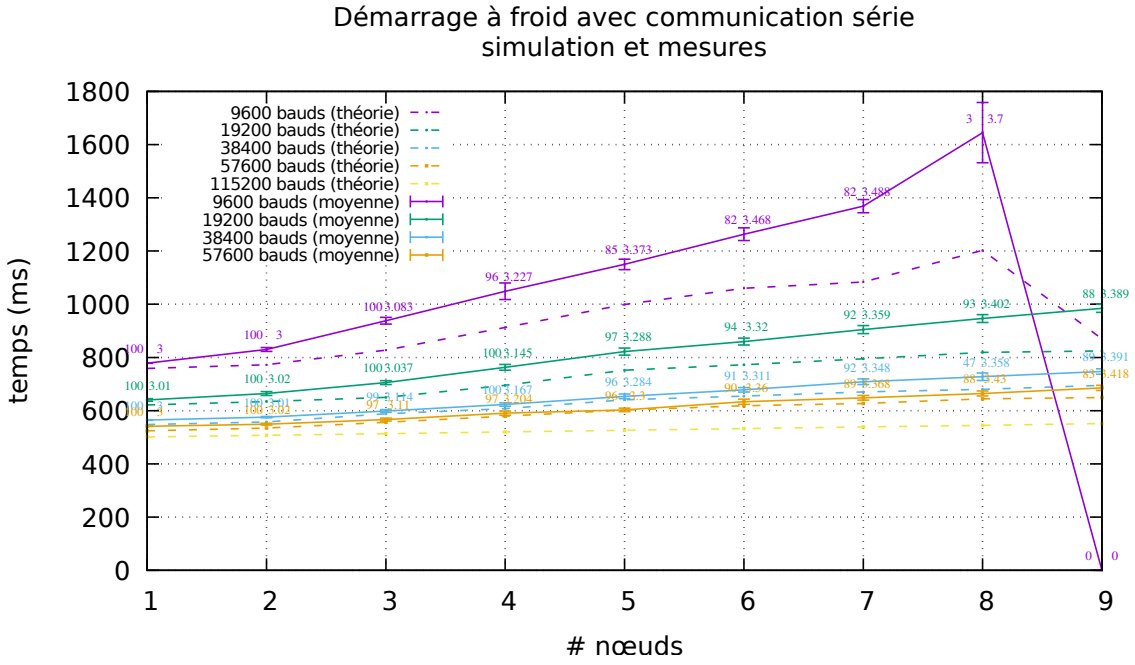


FIGURE C.2 – Démarrage à froid de l'architecture CASAN-S. Les courbes représentent à la fois les valeurs théoriques calculées (en traits discontinus) et les mesures (courbes pleines) avec différentes vitesses de communication série. Chaque point de mesure a deux valeurs : le taux de succès de démarrage du réseau (c'est-à-dire le taux de nœuds qui arrivent à terminer leur communication avec le maître) et la moyenne du nombre de messages envoyés (et acquittés) par les nœuds ayant réussi leur démarrage.

Les courbes théoriques prennent en compte les calculs des temps de transmission et d'attente sur le réseau 802.15.4, ainsi que les temps de transferts sur la liaison série. En prenant uniquement ces temps en compte, les courbes théoriques auraient été lisses, ce qui peut être le cas dans un réseau déployé sans appareil tel que le périphérique IEEE 802.15.4 où les messages sont traités directement, sans retransmission sur une liaison plus lente. Le problème avec cet appareil est qu'il perd des paquets dès lors qu'il n'arrive plus à les transférer à l'hôte aussi vite qu'ils sont envoyés sur le réseau 802.15.4. Afin de montrer la corrélation entre les durées effectives des démarrages et les retards engendrés sur le réseau, nous avons ajouté aux courbes théoriques les retransmissions sur le réseau. Les retransmissions considérées sont les messages transmis et ayant été acquittés par le périphérique IEEE 802.15.4. Pour chaque message supplémentaire transmis, nous comptons le temps d'attente et de transfert sur le réseau 802.15.4, le temps de transfert sur le lien série, ainsi qu'un temps d'attente aléatoire avant l'envoi d'une retransmission (temps défini par le protocole CASAN-S, et non lié au standard 802.15.4). Pour finir, les courbes théoriques prennent en compte les temps de traitement des opérations cryptographiques.

La figure C.2 présente des temps de démarrage d'un réseau en variant la vitesse de communication du port série. CASAN-S est configuré pour retransmettre des messages dans un intervalle de temps aléatoire, comme décrit au tableau C.1, afin d'augmenter le taux de nœuds terminant leur connexion au réseau. Le temps d'attente avant retransmission est choisi de manière aléatoire parmi les intervalles donnés, mais la graine utilisée pour générer cet aléatoire est le numéro du nœud, c'est-à-dire que l'aléatoire est reproductible. Sachant le taux de retransmission sur chaque nœud ainsi que les temps de retransmission grâce à l'aléatoire reproductible, il nous a été possible de créer des courbes théoriques prenant en compte ces durées avant retransmission. Nous pouvons voir sur la figure C.2 une corrélation entre les courbes de mesure et les courbes théoriques où les temps de retransmission ont été pris en compte.

Pour résumer, les courbes théoriques prennent en compte les temps induits par le réseau 802.15.4 et la liaison série, le temps de traitement des opérations cryptographiques et également des erreurs de communication (pertes sur le lien série, non sur le réseau 802.15.4 puisque les messages sont acquittés). Ainsi les courbes dessinent une tendance très proche de celle des mesures réelles.

L'écart entre les courbes théoriques et les mesures réelles s'explique par plusieurs critères non pris en compte. Premièrement, les durées d'attente avant émission sur le réseau 802.15.4 qui augmentent avec le nombre de nœuds : il est considéré que chaque nœud attend 1.12 ms en moyenne avant d'envoyer un message, ce qui correspond au temps d'attente en moyenne pour l'algorithme CSMA-CA lors du premier essai d'émission (voir section 2.2.3), peu importe le nombre de nœuds dans le réseau. Ensuite, des erreurs ne sont pas prises en compte, comme les transmissions sans acquittement (ce qui arrive quand on dépasse les capacités du périphérique IEEE 802.15.4). Et enfin les durées induites par des messages non pris en compte dans le résultat final, c'est-à-dire des paquets liés à des démarrages se terminant au delà de la limite imposée de 5 secondes.

Annexe D

Conversion de bauds à octets par seconde

L'unité `baud` se converti en `octets/s` en prenant en compte les 2 bits de START et STOP. Soit X une vitesse en bauds, $V_{\text{octets/s}} = \frac{X}{8+2\text{bits}} = \frac{X}{10}$. Le tableau D.1 donne les différentes vitesses utilisées sur le périphérique IEEE 802.15.4 lors des expérimentations et leur correspondance en octets/s.

Vitesse en bauds	Vitesse en octets par seconde
9600	960
19200	1920
38400	3840
57600	5760
115200	11520

TABLE D.1 – Conversion de vitesses. Les bits de START et STOP sont pris en compte.

Bibliographie

[1] D. Evans, « The Internet of Things : How the Next Evolution of the Internet is Changing Everything », *Cisco Internet Business Solutions Group (IBSG)*, t. 1, p. 1–11, jan. 2011.

[2] *The Internet of Things Infographic*. 2011. adresse : https://blogs.cisco.com/diversity/the-internet-of-things-infographic.

[3] I. Akyildiz, W. Su, Y. Sankarasubramaniam et E. Cayirci, « Wireless Sensor Networks : a Survey », *Computer Networks*, t. 38, nº 4, p. 393 –422, 2002, issn : 1389-1286. doi : https://doi.org/10.1016/S1389-1286(01)00302-4. adresse : http://www.sciencedirect.com/science/article/pii/S1389128601003024.

[4] K. Langendoen, A. Baggio et O. Visser, « Murphy loves potatoes : experiences from a pilot sensor network deployment in precision agriculture », in *Proceedings 20th IEEE International Parallel Distributed Processing Symposium*, 2006, 8 pp.–. doi : 10.1109/IPDPS.2006.1639412.

[5] O. Chipara, C. Lu, T. Bailey et G.-C. Roman, « Reliable Clinical Monitoring using Wireless Sensor Networks : Experiences in a Step-down Hospital Unit », oct. 2010, p. 155–168. doi : 10.1145/1869983.1869999.

[6] X. Mao, X. Miao, Y. He, X.-Y. Li et Y. Liu, « CitySee : Urban CO2 monitoring with sensors », *Proceedings - IEEE INFOCOM*, p. 1611–1619, mar. 2012. doi : 10.1109/INFCOM.2012.6195530.

[7] B. O'Flynn, R. Martinez, J. Cleary, C. Slater, F. Regan, D. Diamond et H. Murphy, « SmartCoast : A Wireless Sensor Network for Water Quality Monitoring », nov. 2007, p. 815 –816, isbn : 0-7695-3000-1. doi : 10.1109/LCN.2007.34.

[8] G. Simon, M. Maróti, A. Ledeczi, G. Balogh, B. Kusy, A. Nádas, G. Pap, J. Sallai et K. Frampton, « Sensor network-based countersniper system », jan. 2004, p. 1–12. doi : 10.1145/1031495.1031497.

[9] C. Bormann, M. Ersue et A. Keranen, *Terminology for Constrained-Node Networks*, RFC 7228 (Informational), Internet Engineering Task Force, mai 2014. adresse : http://www.ietf.org/rfc/rfc7228.txt.

[10] N. Baccour, A. Koubâa, L. Mottola, M. A. Zúñiga, H. Youssef, C. A. Boano et M. Alves, « Radio Link Quality Estimation in Wireless Sensor Networks : A Survey », *ACM Trans. Sen. Netw.*, t. 8, nº 4, 34 :1–34 :33, sept. 2012, issn : 1550-4859. doi : 10.1145/2240116.2240123. adresse : http://doi.acm.org/10.1145/2240116.2240123.

[11] IEEE, *Ieee 802.15.4 datasheet*, http ://standards.ieee.org/getieee802/download/802.15.4-2006.pdf, 2006.

[12] N. Salman, I. Rasool et A. H. Kemp, « Overview of the IEEE 802.15.4 standards family for Low Rate Wireless Personal Area Networks », in *2010 7th International Symposium on Wireless Communication Systems*, 2010, p. 701–705. doi : 10.1109/ISWCS.2010.5624516.

[13] I. Ramachandran, A. K. Das et S. Roy, « Analysis of the Contention Access Period of IEEE 802.15.4 MAC », *ACM Trans. Sen. Netw.*, t. 3, n° 1, mar. 2007, issn : 1550-4859. doi : 10.1145/1210669.1210673. adresse : http://doi.acm.org/10.1145/1210669.1210673.

[14] Y. Li, B. Bakkaloglu et C. Chakrabarti, « A Comprehensive Energy Model and Energy-Quality Evaluation of Wireless Transceiver Front-ends », in *IEEE Workshop on Signal Processing Systems Design and Implementation, 2005.*, 2005, p. 262–267. doi : 10.1109/SIPS.2005.1579876.

[15] G. Montenegro, N. Kushalnagar, J. Hui et D. Culler, *Transmission of IPv6 Packets over IEEE 802.15.4 Networks*, RFC 4944 (Proposed Standard), Internet Engineering Task Force, sept. 2007. adresse : http://www.ietf.org/rfc/rfc4944.txt.

[16] J. Hui et P. Thubert, *Compression format for ipv6 datagrams over ieee 802.15.4-based networks*, RFC 6282 (Proposed Standard), Internet Engineering Task Force, sept. 2011. adresse : http://www.ietf.org/rfc/rfc6282.txt.

[17] C. Bormann, *6LoWPAN-GHC : Generic Header Compression for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)*, RFC 7400 (Proposed Standard), Internet Engineering Task Force, nov. 2014. adresse : http://www.ietf.org/rfc/rfc7400.txt.

[18] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur et R. Alexander, *RPL : IPv6 Routing Protocol for Low-Power and Lossy Networks*, RFC 6550 (Proposed Standard), Internet Engineering Task Force, mar. 2012. adresse : http://www.ietf.org/rfc/rfc6550.txt.

[19] OASIS. (2015). Message Queuing Telemetry Transport (MQTT), adresse : https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/errata01/os/mqtt-v3.1.1-errata01-os-complete.html (visité le 30/12/2016).

[20] Z. Shelby, K. Hartke et C. Bormann, *The Constrained Application Protocol (CoAP)*, RFC 7252 (Proposed Standard), Internet Engineering Task Force, juin 2014. adresse : http://www.ietf.org/rfc/rfc7252.txt.

[21] J. Jimenez, J. Lopez-Vega, J. Maenpaa et G. Camarillo, *A constrained application protocol (coap) usage for resource location and discovery (reload)*, RFC 7650 (Proposed Standard), Internet Engineering Task Force, sept. 2015. adresse : http://www.ietf.org/rfc/rfc7650.txt.

[22] A. Castellani, S. Loreto, A. Rahman, T. Fossati et E. Dijk, « Guidelines for mapping implementations : http to the constrained application protocol (coap) », RFC Editor, RFC 8075, 2017.

[23] M. Koster, A. Keränen et J. Jimenez, « Publish-subscribe broker for the constrained application protocol (coap) », Internet Engineering Task Force, Internet-Draft draft-ietf-core-coap-pubsub-06, jan. 2019, Work in Progress, 24 p. adresse : https://datatracker.ietf.org/doc/html/draft-ietf-core-coap-pubsub-06.

[24] D. McGrew et D. Bailey, *Aes-ccm cipher suites for transport layer security (tls)*, RFC 6655 (Proposed Standard), Internet Engineering Task Force, juil. 2012. adresse : http://www.ietf.org/rfc/rfc6655.txt.

[25] J. Callas, L. Donnerhacke, H. Finney, D. Shaw et R. Thayer, *OpenPGP Message Format*, RFC 4880 (Proposed Standard), Internet Engineering Task Force, nov. 2007. adresse : http://www.ietf.org/rfc/rfc4880.txt.

[26] S. Bellovin et R. Housley, *Guidelines for Cryptographic Key Management*, RFC 4107 (Best Current Practice), Internet Engineering Task Force, juin 2005. adresse : http://www.ietf.org/rfc/rfc4107.txt.

[27] A. Langley, M. Hamburg et S. Turner, *Elliptic Curves for Security*, RFC 7748 (Informational), Internet Engineering Task Force, jan. 2016. adresse : http://www.ietf.org/rfc/rfc7748.txt.

[28] S. Blake Wilson, N. Bolyard, V. Gupta, C. Hawk et B. Moeller, *Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)*, RFC 4492 (Informational), Internet Engineering Task Force, mai 2006. adresse : http://www.ietf.org/rfc/rfc4492.txt.

[29] E. Barker, « Recommendation for key management - part 1 : general (revised) », in *NIST Special Publication 800-57 Part 1, revision 4*, 2016.

[30] A. Liu et P. Ning, « TinyECC : a configurable library for elliptic curve cryptography in wireless sensor networks », in *Proceedings of the 7th International Conference on Information Processing in Sensor Networks*, sér. IPSN '08, Washington, DC, USA : IEEE Computer Society, 2008, p. 245–256. adresse : http://dx.doi.org/10.1109/IPSN.2008.47.

[31] J. Schaad, *CBOR Object Signing and Encryption (COSE)*, RFC 8152 (Proposed Standard), RFC, Fremont, CA, USA : RFC Editor, juil. 2017. doi : 10.17487/RFC8152. adresse : https://www.rfc-editor.org/rfc/rfc8152.txt.

[32] C. Bormann et P. Hoffman, *Concise Binary Object Representation (CBOR)*, RFC 7049 (Proposed Standard), Internet Engineering Task Force, oct. 2013. adresse : http://www.ietf.org/rfc/rfc7049.txt.

[33] R. Barnes, *Use Cases and Requirements for JSON Object Signing and Encryption (JOSE)*, RFC 7165 (Informational), Internet Engineering Task Force, avr. 2014. adresse : http://www.ietf.org/rfc/rfc7165.txt.

[34] R. Housley, *Guidelines for Cryptographic Algorithm Agility and Selecting Mandatory-to-Implement Algorithms*, RFC 7696 (Best Current Practice), RFC, Fremont, CA, USA : RFC Editor, nov. 2015. doi : 10.17487/RFC7696. adresse : https://www.rfc-editor.org/rfc/rfc7696.txt.

[35] E. Rescorla et N. Modadugu, *Datagram Transport Layer Security Version 1.2*, RFC 6347 (Proposed Standard), Internet Engineering Task Force, jan. 2012. adresse : http://www.ietf.org/rfc/rfc6347.txt.

[36] H. Tschofenig et T. Fossati, *Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things*, RFC 7925 (Proposed Standard), Internet Engineering Task Force, juil. 2016. adresse : http://www.ietf.org/rfc/rfc7925.txt.

[37] A. Rahman et E. Dijk, *Group Communication for the Constrained Application Protocol (CoAP)*, RFC 7390 (Experimental), Internet Engineering Task Force, oct. 2014. adresse : http://www.ietf.org/rfc/rfc7390.txt.

[38] Matthew Green, *Why I hate CBC-MAC*. adresse : https://blog.cryptographyengineering.com/2013/02/15/why-i-hate-cbc-mac/.

[39] Yong Wang, G. Attebury et B. Ramamurthy, « A survey of Security issues in Wireless Sensor Networks », *Communications Surveys Tutorials, IEEE*, t. 8, n° 2, p. 2–23, 2006, issn : 1553-877X. doi : 10.1109/COMST.2006.315852.

[40] A. Mayzaud, R. Badonnel et I. Chrisment, « A Taxonomy of Attacks in RPL-based Internet of Things », *International Journal of Network Security*, t. 18, n° 3, p. 459 –473, mai 2016. adresse : https://hal.inria.fr/hal-01207859.

[41] B. Sarikaya, M. Sethi et D. Garcia-Carillo, « Secure iot bootstrapping : a survey », Internet Engineering Task Force, Internet-Draft draft-sarikaya-t2trg-sbootstrapping-05, sept. 2018, Work in Progress, 21 p. adresse : https://datatracker.ietf.org/doc/html/draft-sarikaya-t2trg-sbootstrapping-05.

[42] ITU, *X.805 : Security Architecture for Systems Providing End-to-End Communications*, 2003. adresse : http://www.itu.int/rec/T-REC-X.805-200310-I/en.

[43] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson et H. Levkowetz, *Extensible authentication protocol (eap)*, RFC 3748 (Proposed Standard), Updated by RFCs 5247, 7057, Internet Engineering Task Force, juin 2004. adresse : http://www.ietf.org/rfc/rfc3748.txt.

[44] H. Haverinen et J. Salowey, *Extensible authentication protocol method for global system for mobile communications (gsm) subscriber identity modules (eap-sim)*, RFC 4186 (Informational), Internet Engineering Task Force, jan. 2006. adresse : http://www.ietf.org/rfc/rfc4186.txt.

[45] J. Arkko et H. Haverinen, *Extensible authentication protocol method for 3rd generation authentication and key agreement (eap-aka)*, RFC 4187 (Informational), Updated by RFC 5448, Internet Engineering Task Force, jan. 2006. adresse : http://www.ietf.org/rfc/rfc4187.txt.

[46] J. Arkko, V. Lehtovirta et P. Eronen, *Improved Extensible Authentication Protocol Method for 3rd Generation Authentication and Key Agreement (EAP-AKA')*, RFC 5448 (Informational), Internet Engineering Task Force, mai 2009. adresse : http://www.ietf.org/rfc/rfc5448.txt.

[47] M. Nystroem, *The eap protected one-time password protocol (eap-potp)*, RFC 4793 (Informational), Internet Engineering Task Force, fév. 2007. adresse : http://www.ietf.org/rfc/rfc4793.txt.

[48] A. Palekar, S. Josefsson, D. Simon et G. Zorn, « Protected eap protocol (peap) version 2 », Internet Engineering Task Force, Internet-Draft draft-josefsson-pppext-eap-tls-eap-10, oct. 2004, Work in Progress, 87 p. adresse : https://datatracker.ietf.org/doc/html/draft-josefsson-pppext-eap-tls-eap-10.

[49] P. Funk et S. Blake-Wilson, *Extensible authentication protocol tunneled transport layer security authenticated protocol version 0 (eap-ttlsv0)*, RFC 5281 (Informational), Internet Engineering Task Force, août 2008. adresse : http://www.ietf.org/rfc/rfc5281.txt.

[50] Y. Sheffer, G. Zorn, H. Tschofenig et S. Fluhrer, *An eap authentication method based on the encrypted key exchange (eke) protocol*, RFC 6124 (Informational), Internet Engineering Task Force, fév. 2011. adresse : http://www.ietf.org/rfc/rfc6124.txt.

[51] F. Bersani et H. Tschofenig, *The eap-psk protocol : a pre-shared key extensible authentication protocol (eap) method*, RFC 4764 (Experimental), Internet Engineering Task Force, jan. 2007. adresse : http://www.ietf.org/rfc/rfc4764.txt.

[52] C. Rigney, S. Willens, A. Rubens et W. Simpson, *Remote authentication dial in user service (radius)*, RFC 2865 (Draft Standard), Updated by RFCs 2868, 3575, 5080, 6929, Internet Engineering Task Force, juin 2000. adresse : http://www.ietf.org/rfc/rfc2865.txt.

[53] C. Rigney, *Radius accounting*, RFC 2866 (Informational), Updated by RFCs 2867, 5080, 5997, Internet Engineering Task Force, juin 2000. adresse : http://www.ietf.org/rfc/rfc2866.txt.

[54] V. Fajardo, J. Arkko, J. Loughney et G. Zorn, *Diameter base protocol*, RFC 6733 (Proposed Standard), Updated by RFC 7075, Internet Engineering Task Force, oct. 2012. adresse : http://www.ietf.org/rfc/rfc6733.txt.

[55] Thorsten Dahm, Andrej Ota, dcmgash@cisco.com, David Carrel et Lol Grant, « The TACACS+ Protocol », Internet Engineering Task Force, Internet-Draft draft-ietf-opsawg-tacacs-12, déc. 2018, Work in Progress, 44 p. adresse : https://datatracker.ietf.org/doc/html/draft-ietf-opsawg-tacacs-12.

[56] S. Gerdes, O. Bergmann et C. Bormann, *Delegated CoAP Authentication and Authorization Framework (DCAF)*, Working Draft, Internet-Draft, Internet Engineering Task Force, avr. 2015.

[57] L. Seitz, S. Gerdes, G. Selander, M. Mani et S. Kumar, *Use Cases for Authentication and Authorization in Constrained Environments*, RFC 7744 (Informational), Internet Engineering Task Force, jan. 2016. adresse : http://www.ietf.org/rfc/rfc7744.txt.

[58] Ludwig Seitz, Goeran Selander, Erik Wahlstroem, Samuel Erdtman et Hannes Tschofenig, « Authentication and Authorization for Constrained Environments (ACE) », IETF Secretariat, Internet-Draft draft-ietf-ace-oauth-authz-09, 2017, http://www.ietf.org/internet-drafts/draft-ietf-ace-oauth-authz-09.txt. adresse : http://www.ietf.org/internet-drafts/draft-ietf-ace-oauth-authz-09.txt.

[59] Goeran Selander, John Mattsson et Francesca Palombini, « Ephemeral Diffie-Hellman Over COSE (EDHOC) », IETF Secretariat, Internet-Draft draft-selander-ace-cose-ecdhe-08, 2018, http://www.ietf.org/internet-drafts/draft-selander-ace-cose-ecdhe-08.txt. adresse : http://www.ietf.org/internet-drafts/draft-selander-ace-cose-ecdhe-08.txt.

[60] Ludwig Seitz, Francesca Palombini, Martin Gunnarsson et Goeran Selander, « OSCORE profile of the Authentication and Authorization for Constrained Environments Framework », IETF Secretariat, Internet-Draft draft-ietf-ace-oscore-profile-01, 2018, http://www.ietf.org/internet-drafts/draft-ietf-ace-oscore-profile-01.txt. adresse : http://www.ietf.org/internet-drafts/draft-ietf-ace-oscore-profile-01.txt.

[61] Stefanie Gerdes, Olaf Bergmann, Carsten Bormann, Goeran Selander et Ludwig Seitz, « Datagram Transport Layer Security (DTLS) Profiles for Authentication and Authorization for Constrained Environments (ACE) », IETF Secretariat, Internet-Draft draft-ietf-ace-dtls-authorize-02, 2017, http://www.ietf.org/internet-drafts/draft-ietf-ace-dtls-authorize-02.txt. adresse : http://www.ietf.org/internet-drafts/draft-ietf-ace-dtls-authorize-02.txt.

[62] John Mattsson, « Using Transport Layer Security (TLS) to Secure OSCORE », IETF Secretariat, Internet-Draft draft-mattsson-ace-tls-oscore-00, 2017, http://www.ietf.org/internet-drafts/draft-mattsson-ace-tls-oscore-00.txt. adresse : http://www.ietf.org/internet-drafts/draft-mattsson-ace-tls-oscore-00.txt.

[63] S. Aragon, M. Tiloca et S. Raza, « IPsec profile of ACE », IETF Secretariat, Internet-Draft draft-aragon-ace-ipsec-profile-01, 2017, http://www.ietf.org/internet-drafts/draft-aragon-ace-ipsec-profile-01.txt. adresse : http://www.ietf.org/internet-drafts/draft-aragon-ace-ipsec-profile-01.txt.

[64] R. Lopez et D. Garcia, « EAP-based Authentication Service for CoAP », IETF Secretariat, Internet-Draft draft-marin-ace-wg-coap-eap-06, 2017, http://www.ietf.org/internet-drafts/draft-marin-ace-wg-coap-eap-06.txt. adresse : http://www.ietf.org/internet-drafts/draft-marin-ace-wg-coap-eap-06.txt.

[65] Cigdem Sengul, Anthony Kirby et Paul Fremantle, « MQTT-TLS profile of ACE », IETF Secretariat, Internet-Draft draft-sengul-ace-mqtt-tls-profile-02, 2018, http://www.ietf.org/internet-drafts/draft-sengul-ace-mqtt-tls-profile-02.txt. adresse : http://www.ietf.org/internet-drafts/draft-sengul-ace-mqtt-tls-profile-02.txt.

[66] C. Neuman, T. Yu, S. Hartman et K. Raeburn, *The Kerberos Network Authentication Service (v5)*, RFC 4120 (Proposed Standard), Internet Engineering Task Force, juil. 2005. adresse : http://www.ietf.org/rfc/rfc4120.txt.

[67] Mališa Vučinić, Bernard Tourancheau, Franck Rousseau, Andrzej Duda, Laurent Damon et Roberto Guizzetti, « OSCAR : Object Security Architecture for the Internet of Things », *Ad Hoc Networks*, t. 32, p. 3 –16, 2015, Internet of Things security and privacy : design methods and optimization. adresse : http://www.sciencedirect.com/science/article/pii/S1570870514003126.

[68] G. Selander, J. Mattsson, F. Palombini et L. Seitz, « Object Security for Constrained RESTful Environments (OSCORE) », IETF Secretariat, Internet-Draft draft-ietf-core-object-security-14, 2018, http://www.ietf.org/internet-drafts/draft-ietf-core-object-security-14.txt. adresse : http://www.ietf.org/internet-drafts/draft-ietf-core-object-security-14.txt.

[69] S. Kent et K. Seo, *Security Architecture for the Internet Protocol*, RFC 4301 (Proposed Standard), Internet Engineering Task Force, déc. 2005. adresse : http://www.ietf.org/rfc/rfc4301.txt.

[70] J. Granjal, R. Silva, E. Monteiro, J. Sa Silva et F. Boavida, « Why is IPsec a Viable Option for Wireless Sensor Networks », in *2008 5th IEEE International Conference on Mobile Ad Hoc and Sensor Systems*, 2008, p. 802–807. doi : 10.1109/MAHSS.2008.4660130.

[71] S. Kent, *IP Encapsulating Security Payload (ESP)*, RFC 4303 (Proposed Standard), Internet Engineering Task Force, déc. 2005. adresse : http://www.ietf.org/rfc/rfc4303.txt.

[72] D. McGrew et P. Hoffman, *Cryptographic Algorithm Implementation Requirements and Usage Guidance for Encapsulating Security Payload (ESP) and Authentication Header (AH)*, RFC 7321 (Proposed Standard), Internet Engineering Task Force, août 2014. adresse : http://www.ietf.org/rfc/rfc7321.txt.

[73] S. Kent, *IP Authentication Header*, RFC 4302 (Proposed Standard), Internet Engineering Task Force, déc. 2005. adresse : http://www.ietf.org/rfc/rfc4302.txt.

[74] C. Kaufman, P. Hoffman, Y. Nir, P. Eronen et T. Kivinen, *Internet Key Exchange Protocol Version 2 (ikev2)*, Internet Engineering Task Force, oct. 2014. adresse : http://www.ietf.org/rfc/rfc7296.txt.

[75] J. Schiller, *Cryptographic Algorithms for Use in the Internet Key Exchange Version 2 (IKEv2)*, RFC 4307 (Proposed Standard), Internet Engineering Task Force, déc. 2005. adresse : http://www.ietf.org/rfc/rfc4307.txt.

[76] E. Jankiewicz, J. Loughney et T. Narten, *IPv6 Node Requirements*, RFC 6434 (Informational), Internet Engineering Task Force, déc. 2011. adresse : http://www.ietf.org/rfc/rfc6434.txt.

[77] T. Ylonen et C. Lonvick, *The Secure Shell (SSH) Protocol Architecture*, RFC 4251 (Proposed Standard), Internet Engineering Task Force, jan. 2006. adresse : http://www.ietf.org/rfc/rfc4251.txt.

[78] ——, *The Secure Shell (SSH) Transport Layer Protocol*, RFC 4253 (Proposed Standard), Internet Engineering Task Force, jan. 2006. adresse : http://www.ietf.org/rfc/rfc4253.txt.

[79] ——, *The Secure Shell (SSH) Authentication Protocol*, RFC 4252 (Proposed Standard), Internet Engineering Task Force, jan. 2006. adresse : http://www.ietf.org/rfc/rfc4252.txt.

[80] ——, *The Secure Shell (SSH) Connection Protocol*, RFC 4254 (Proposed Standard), Internet Engineering Task Force, jan. 2006. adresse : http://www.ietf.org/rfc/rfc4254.txt.

[81] K. Igoe et J. Solinas, *AES Galois Counter Mode for the Secure Shell Transport Layer Protocol*, RFC 5647 (Informational), Internet Engineering Task Force, août 2009. adresse : http://www.ietf.org/rfc/rfc5647.txt.

[82] A. Liu et P. Ning, « TinyECC : A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks », in *Proceedings of the 7th International Conference on Information Processing in Sensor Networks*, sér. IPSN '08, Washington, DC, USA : IEEE Computer Society, 2008, p. 245–256, isbn : 978-0-7695-3157-1. doi : 10.1109/IPSN.2008.47. adresse : http://dx.doi.org/10.1109/IPSN.2008.47.

[83] M. Vučinić, B. Tourancheau, T. Watteyne, F. Rousseau, A. Duda, R. Guizzetti et L. Damon, « DTLS Performance in Duty-Cycled Networks », in *International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC - 2015)*, Hong-Kong, China : IEEE, 2015.

[84] H. Tschofenig et T. Fossati, « TLS/DTLS Profiles for the Internet of Things », IETF Secretariat, Internet-Draft draft-ietf-dice-profile-17.txt, oct. 2015. adresse : http://www.rfc-editor.org/internet-drafts/draft-ietf-dice-profile-17.txt.

[85] S. Raza, H. Shafagh, K. Hewage, H. Rene et T. Voigt, « Lithe : Lightweight Secure CoAP for the Internet of Things », *IEEE Sensors Journal*, t. 13, nº 10, p. 3711–3720, 2013.

[86] C. Bormann, C. Burmeister, M. Degermark, H. Fukushima, H. Hannu, L.-E. Jonsson, R. Hakenberg, T. Koren, K. Le, Z. Liu, A. Martensson, A. Miyazaki, K. Svanbro, T. Wiebke, T. Yoshimura et H. Zheng, *Robust header compression (rohc) : framework and four profiles : rtp, udp, esp, and uncompressed*, RFC 3095 (Proposed Standard), Updated by RFCs 3759, 4815, Internet Engineering Task Force, juil. 2001. adresse : http://www.ietf.org/rfc/rfc3095.txt.

[87] G. Pelletier, K. Sandlund, L.-E. Jonsson et M. West, *Robust header compression (rohc) : a profile for tcp/ip (rohc-tcp)*, RFC 4996 (Proposed Standard), Obsoleted by RFC 6846, Internet Engineering Task Force, juil. 2007. adresse : http://www.ietf.org/rfc/rfc4996.txt.

[88] G. Pelletier et K. Sandlund, *Robust header compression version 2 (rohcv2) : profiles for rtp, udp, ip, esp and udp-lite*, RFC 5225 (Proposed Standard), Internet Engineering Task Force, avr. 2008. adresse : http://www.ietf.org/rfc/rfc5225.txt.

[89] K. Sandlund, G. Pelletier et L.-E. Jonsson, *The robust header compression (rohc) framework*, RFC 5795 (Proposed Standard), Internet Engineering Task Force, mar. 2010. adresse : http://www.ietf.org/rfc/rfc5795.txt.

[90] G. Pelletier, K. Sandlund, L.-E. Jonsson et M. West, *Robust header compression (rohc) : a profile for tcp/ip (rohc-tcp)*, RFC 6846 (Proposed Standard), Internet Engineering Task Force, jan. 2013. adresse : http://www.ietf.org/rfc/rfc6846.txt.

[91] René Hummen, Hossein Shafagh, Shahid Raza, Thiemo Voigt et Klaus Wehrle, « Delegation-based Authentication and Authorization for the IP-based Internet of Things », in *Proceedings of the 11th IEEE International Conference on Sensing, Communication, and Networking (SECON'14). Singapore*, juin 2014, p. 284–292.

[92] P. David et T. Noël, « CASAN : A New Communication Architecture for Sensors Based on CoAP », in *12th IEEE International Conference on Networking, Sensing and Control*, IEEE, 2015.

[93] Z. Shelby, *Constrained restful environments (core) link format*, RFC 6690 (Proposed Standard), Internet Engineering Task Force, août 2012. adresse : http://www.ietf.org/rfc/rfc6690.txt.

[94] Z. Shelby, M. Koster, C. Bormann, P. van der Stok et C. Amsuess, « Core resource directory », IETF Secretariat, Internet-Draft draft-ietf-core-resource-directory-19, 2019, http://www.ietf.org/internet-drafts/draft-ietf-core-resource-directory-19.txt. adresse : http://www.ietf.org/internet-drafts/draft-ietf-core-resource-directory-19.txt.

[95] P. Pittoli, P. David et T. Noël, « DTLS Improvements for Fast Handshake and Bigger Payload in Constrained Environments », in *Ad-hoc, Mobile, and Wireless Networks*, N. Mitton, V. Loscri et A. Mouradian, éds., Springer International Publishing, 2016, isbn : 978-3-319-40509-4.

[96] C. Bormann et Z. Shelby, *Block-wise transfers in the constrained application protocol (coap)*, RFC 7959 (Proposed Standard), Internet Engineering Task Force, août 2016. adresse : http://www.ietf.org/rfc/rfc7959.txt.

[97] Atmel, *ATmega128RFA1 datasheet*, 2014. adresse : http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-8266-MCU_Wireless-ATmega128RFA1_Datasheet.pdf.

[98] Communauté, *Contiki OS*, 2014. adresse : http://contiki-os.org.

[99] R. Navas, B. Gaultier et L. Toutain, *Bibliothèque logicielle PicoIPv6*. adresse : https://github.com/telecombretagne/Arduino-pIPv6Stack.

[100] Philippe Pittoli, Pierre David et Thomas Noël, « Security Architectures in Constrained Environments : a Survey », *Ad Hoc Networks*, t. 79, p. 112 –132, 2018, issn : 1570-8705. doi : https://doi.org/10.1016/j.adhoc.2018.06.001. adresse : http://www.sciencedirect.com/science/article/pii/S1570870518303007.

[101] M. Collina, G. Corazza et A. Vanelli Coralli, « Introducing the QESt broker : scaling the IoT by bridging MQTT and REST », sept. 2012, p. 36–41, isbn : 978-1-4673-2566-0. doi : 10.1109/PIMRC.2012.6362813.

[102] Digi, *Documentation du stick Digi IEEE 802.15.4 USB*. adresse : https://www.digi.com/products/xbee-rf-solutions/boxed-rf-modems-adapters/xstick.

Philippe PITTOLI

Influence d’une architecture maître-esclave dans les problématiques de sécurité de l’Internet des Objets

Résumé

Insérer votre résumé en français suivi des mots-clés

L’Internet des objets est un principe selon lequel des clients sur Internet peuvent contacter des objets intelligents de notre quotidien, comme des capteurs de température d'une pièce ou des ampoules connectées.

Ces objets sont contraints en mémoire, en capacité de calcul et aussi en capacité de communication (taille des paquets, médium partagé).

Le travail effectué se focalise sur des problématiques liées à ces contraintes.

Lorsqu'un client souhaite envoyer une commande à un objet, il a le choix de s'y connecter directement (architecture bout-en-bout) ou de se connecter à une passerelle réseau (non contrainte en mémoire et en calculs) qui jouera le rôle d’intermédiaire entre les clients et les objets (architecture maître-esclave). Le travail effectué consiste à comprendre les différences entre ces deux architectures et leur viabilité dans des réseaux de l'Internet des Objets.

Mots clés : Internet des Objets, Sécurité, Protocoles, Réseau

Résumé en anglais

Insérer votre résumé en anglais suivi des mots-clés

The Internet of things is a network design where "things" are connected to the Internet, such as thermometers or lights.

These objects are constrained in memory, computational capacity and communication (packet size, shared medium).

The thesis is focused on issues around those constraints.

A client willing to send a request to an object may either establish a direct connection to the object (end-to-end architecture) or establish a connection to the network gateway, which is not constrained in memory or computation capabilities, and will be used as a broker between clients and objects (master-slave architecture). This purpose of the thesis is to understand and to spotlight the differences between those two kinds of architectures and to determine their viability in an IoT context.

Keywords : Internet of Things, Security, Protocols, Networks