Intégration sécurisée d'objets connectés dans les applications pervasives

Arthur Desuert

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

École doctorale : MSTII - Mathématiques, Sciences et technologies de l'information, Informatique
Spécialité : Informatique
Unité de recherche : Laboratoire de conception et d'intégration des systèmes

Intégration sécurisée d'objets connectés dans les applications pervasives

Secure internet-of-things integration into pervasive applications

Présentée par :

Arthur DESUERT

Direction de thèse :

Stéphanie CHOLLET MAITRE DE CONFERENCES HDR, GRENOBLE INP	Directrice de thèse
David HELY MAITRE DE CONFERENCES HDR, Université Grenoble Alpes	Co-directeur de thèse
Laurent PION Chaire Trust de la Fondation Grenoble INP, Fondation Grenoble INP	Co-encadrant de thèse

Rapporteurs :

PHILIPPE ROOSE
PROFESSEUR DES UNIVERSITES, UNIVERSITE DE PAU ET PAYS DE L'ADOUR
LILIAN BOSSUET
PROFESSEUR DES UNIVERSITES, UNIVERSITE DE SAINT-ETIENNE - JEAN MONNET

Thèse soutenue publiquement le **6 novembre 2023**, devant le jury composé de :

STEPHANIE CHOLLET MAITRE DE CONFERENCES HDR, GRENOBLE INP	Directrice de thèse
PHILIPPE ROOSE PROFESSEUR DES UNIVERSITES, UNIVERSITE DE PAU ET PAYS DE L'ADOUR	Rapporteur
LILIAN BOSSUET PROFESSEUR DES UNIVERSITES, UNIVERSITE DE SAINT-ETIENNE - JEAN MONNET	Rapporteur
PHILIPPE LALANDA PROFESSEUR DES UNIVERSITES, UNIVERSITE GRENOBLE ALPES	Président
YOANN MAUREL MAITRE DE CONFERENCES, UNIVERSITE DE RENNES	Examinateur

Invités :

LAURENT PION
INGENIEUR,
DAVID HELY
MAITRE DE CONFERENCES HDR, UNIVERSITE GRENOBLE ALPES

Remerciements

Je tiens à exprimer ma gratitude à toutes les personnes qui ont participé au bon déroulement et à l'aboutissement de cette thèse : une véritable aventure scientifique, technique et humaine !

Je souhaite d'abord remercier les membres de mon jury. Je remercie Lilian Bossuet et Philippe Roose d'avoir accepté d'être les rapporteurs de ma thèse. Merci également à Philippe Lalanda d'avoir présidé le jury. Enfin, je remercie Yoann Maurel pour sa participation en tant qu'examinateur et pour son implication en tant qu'expert lors de mes comités de suivi.

Comment ne pas remercier mon équipe d'encadrement pour sa disponibilité et son implication durant ces trois années ? Un grand merci à ma directrice de thèse, Stéphanie Chollet, ainsi qu'à Laurent Pion et à David Hély. Une équipe certes exigente, mais avant tout juste et toujours à l'écoute.

Je remercie l'équipe CTSYS, qui m'a accueillie pendant ces trois ans, et plus généralement l'ensemble du personnel scientifique, technique et administratif du LCIS, pour leur contribution à la vie du laboratoire au travers d'événements scientifiques et extra-scientifiques.

Je remercie également la Fondation Grenoble INP et son personnel pour le financement de cette thèse et l'accompagnement bienveillant dont j'ai bénéficié.

Je tiens à adresser des remerciements particuliers à une quinzaine de personnes, rencontrées au début de cette thèse et qui m'ont accompagnées tout au long de cette aventure. Avec cette joyeuse troupe, j'ai trouvé chaussure à mon pied et j'ai découvert bien plus que ce que je cherchais au départ. Un grand merci aux "Talons Cachés".

Merci également à Daphné et à Simba, qui m'ont soutenu à leur manière durant cette aventure.

Pour terminer, je remercie du fond du coeur mes parents, mes grands-parents, mon frère et mes amis proches pour leur soutien et leur présence chaleureuse.

Résumé

Les applications pervasives fournissent des services à des utilisateurs tout en demeurant invisibles, en nécessitant une attention minimale de leur part. En conséquence, une très bonne intégration et gestion de l'hétérogénéité et de la volatilité des appareils est une condition préalable. Un autre sujet très important est la gestion de la cybersécurité (matérielle et logicielle). C'est pour les années à venir une préoccupation majeure de l'informatique pervasive pour continuer son déploiement en toute confiance dans des secteurs tels que la domotique, les villes connectées, l'industrie ou la santé. La cybersécurité doit être aussi transparente et peu contraignante que possible pour les utilisateurs non experts de ces applications. Cela implique la prise en charge d'une large gamme d'objets connectés aux caractéristiques, aux interfaces et aux propriétés de sécurité variées ; allant d'équipements haut de gamme très bien sécurisés à des appareils à faible coût ayant une à la sécurité plus modeste. Elle doit aussi être facile d'accès pour les développeurs d'applications pervasives. Ces derniers doivent, en effet, pouvoir concevoir simplement des applications capables d'interagir de manière sécurisée avec les objets connectés à leur disposition.

Pour répondre à ces besoins, ma thèse propose une nouvelle architecture de plateforme pervasive ayant pour caractéristiques principales l'intégration de la sécurité et sa simplicité d'utilisation. La plateforme agira comme un intermédiaire entre les applications pervasives et les objets connectés, masquant l'hétérogénéité de ces derniers aux applications et gérant la sécurisation des communications. En plus d'avoir une interface homogène sous forme de services, cette plateforme permet aux applications de gérer facilement leur sécurité en évaluant la qualité de la sécurité des services proposés. Cette évaluation est synthétisée sous la forme de différents niveaux de sécurité qui sont associés aux services et que les applications peuvent utiliser pour guider leurs choix d'utilisation d'un objet par rapport à un autre. Du côté des objets connectés, la plateforme vise à permettre la gestion aisée d'un ensemble de protocoles de communication et les configurations de sécurité associées, avec le support de protocoles classiques, comme HTTPS. De plus, notre plateforme permet d'ajouter facilement de nouveaux protocoles pour rester à jour avec les avancées du domaine. Nous avons pu tester cette fonctionnalité avec la seconde contribution de cette thèse : un nouveau protocole léger d'authentification basé sur une technologie de génération d'empreinte liée aux composants électroniques, les Physical Unclonable Functions (PUF). Ce protocole a pour but de sécuriser à faible coût les objets connectés utilisant cette technologie. Pour valider la pertinence et le bon fonctionnement de ces contributions, nous avons élaboré des scénarios de tests concrets basés sur le déploiement d'applications pervasives pour le cas d'usage de la domotique.

Mots-clés : applications pervasives, cybersécurité, objets connectés, fonctions physiques non clonables

Summary

Pervasive applications provide services to users while remaining invisible, requiring minimal attention from them. Accordingly, a good management of the devices heterogeneity and volatility is a prerequisite. Another topic of importance is the management of cybersecurity (hardware and software). It is for the upcoming years a major concern of pervasive computing to continue its deployment in sensitive domains such as home automation, smart cities, 4.0 Industry or smart health. Cybersecurity should be as transparent and unobtrusive as possible for novice applications users. This implies support for a wide range of connected devices with diverse characteristics, interfaces and security properties; ranging from well secured high-tech equipment to low-cost devices with less advanced security features. It must also be simple to use by developers of pervasive applications. They must be able to easily design applications capable of interacting in a secure manner with the connected devices at their disposal.

To meet these needs, my thesis proposes a new pervasive platform architecture whose main characteristics are the integration of security and its ease of use. The platform will act as an intermediary between the pervasive applications and the connected devices, hiding the heterogeneity of the latter from the applications and managing the communication security. In addition to offering a uniform interface using a service approach, the platform allows the applications to easily manage their security by evaluating the security quality of the provided services. This evaluation is synthesized in the form of different levels of security which are associated with the services. The applications can use those security levels to compare the services and to guide their selection. On the connected devices side, the platform aims at allowing the simple management of a set of communication protocols and the associated security configurations, with the support of classic protocols, such as HTTPS for example. Our platform makes it easy to add new protocols to stay up to date. We were able to test this functionality with the second main contribution of this thesis: a new lightweight authentication protocol based on a hardware fingerprint generation technology, the Physical Unclonable Functions (PUF). The protocol goal is to secure the connected devices at an affordable cost. To validate the relevance and proper functioning of these contributions, we designed concrete test scenarios based on the deployment of pervasive applications in home automation use-cases.

Keywords: pervasive applications, cybersecurity, connected devices, physical unclonable functions

TABLE DES MATIÈRES

1 INTRODUCTION

1.1 CONTEXTE

Nous observons la mise en réseau de plus en plus de nos objets du quotidien : il est possible de trouver facilement aujourd'hui des ampoules connectées, des prises électriques pilotables à distance, des caméras IP ; mais, également des climatiseurs ou des volets roulants. Cela s'inscrit dans un mouvement appelé « l'Internet des Objets » ou, en anglais, *Internet of Things* (abrégé en IoT). Ce mouvement et le marché qui lui est associé, n'ont cessé de croître ces dernières années [Cis20]. Tout porte à croire que cette croissance va se poursuivre, pour supporter des cas d'usage toujours plus nombreux dans la domotique, l'industrie ou la santé connectée [MMM21].

De manière générale, les objets connectés ont pour but d'être déployés dans un environnement donné, pour fournir au moins un service. Pour cela, ils vont collecter des données, les traiter et les échanger avec d'autres équipements informatiques via les réseaux de communication à leur disposition. Certains objets peuvent également agir sur l'environnement. Cela est fait en minimisant, autant que possible, le besoin de supervision d'un utilisateur humain. Pour ce dernier, en effet, un des buts premiers de l'achat et de la mise en place d'objets connectés est l'amélioration de sa qualité de vie grâce à l'automatisation de certaines tâches. Dès 1991, Mark Weiser proposait un concept encore plus poussé nommé « informatique pervasive » [Wei91]. Son idée principale était de rendre l'informatique omniprésente mais aussi invisible dans les environnements qui nous entourent : une multitude d'objets connectés travaillant conjointement de manière autonome pour fournir des services ou automatiser des tâches de manière transparente ou, du moins, de la manière la plus naturelle possible pour les utilisateurs, de sorte que ces derniers viendraient à long terme à presque oublier la présence de ces équipements.

Illustrons cette notion avec un cas d'usage concret : la gestion de la température et de la qualité de l'air d'un logement. Actuellement, cet objectif est assuré par un ensemble d'équipements fournissant un service particulier : un chauffage permettant d'élever la température d'une ou plusieurs pièces, une climatisation ayant le comportement inverse et une ventilation mécanique pour assurer le renouvellement de l'air ambiant. Chaque équipement va, souvent, intégrer des fonctionnalités de gestion intelligente, comme le déclenchement sur une plage horaire donnée ou l'utilisation de capteurs externes pour piloter l'allumage ou l'extinction de l'équipement. Cependant, ces équipements étant développés par des entreprises différentes et, parfois, concurrentes, ils sont souvent conçus pour communiquer uniquement avec d'autres équipements développés par la même entreprise ou des entreprises partenaires éventuellement. Ainsi, un utilisateur qui installerait un chauffage, un climatiseur et une ventilation mécanique issues de trois entreprises différentes aurait de fortes chances d'avoir trois systèmes fonctionnant de manière indépendante et sans connaissance des

autres systèmes. Si le système de chauffage requiert un capteur de température et le climatiseur également, mais que ces deux équipements proviennent d'entreprises différentes, alors l'utilisateur devra déployer un capteur dédié pour chaque équipement. A l'échelle d'un logement, bien que cela ne soit pas satisfaisant, il est possible de s'en accommoder. A l'échelle d'un bâtiment, une telle redondance devient problématique, avec les efforts de gestions que cela implique.

Avec l'informatique pervasive, la chaudière, la climatisation et la ventilation exposeraient des interfaces sur le réseau permettant leur pilotage. Ces interfaces seraient ouvertes à tout fournisseur de solutions connectées souhaitant en faire usage. Le logement pourrait alors être équipé de capteurs connectés provenant de diverses entreprises, permettant de relever, par exemple, la température, l'hydrométrie et la quantité de CO_2 à différents endroits du logement. Ces capteurs exposeraient également leurs fonctionnalités en réseau de manière ouverte. Grâce aux données issues des capteurs et à une consigne fournie par l'utilisateur, un système informatique serait en charge de piloter l'utilisation des différents équipements de manière autonome. Grâce à la vision d'ensemble offerte par la mutualisation des données issues des divers capteurs, ce système pourrait contrôler de manière fine les différents équipements, assurer leur collaboration efficace et éviter qu'ils se nuisent. A l'échelle d'un bâtiment intelligent, cela permettrait d'avoir un système unique pour gérer plusieurs dizaines ou centaines d'équipements, sans avoir à choisir une entreprise particulière fournissant cette solution. De surcroît, l'utilisateur n'aurait plus à superviser individuellement chaque équipement. Le système informatique pervasif serait en mesure de détecter les habitudes de ce dernier et de s'y adapter, d'avoir un fonctionnement dédié à la journée et un autre pour la nuit, un pour les périodes de travail et un différent pour les vacances. Il gagnerait ainsi en confort et en sérénité.

Ce cas d'usage illustre bien les avantages que peut apporter l'informatique pervasive : des interactions presque naturelles avec des environnements constitués de nombreux équipements connectés grâce à des systèmes informatiques de gestion invisibles et autonomes ; un gain de temps et d'efforts pour les utilisateurs grâce aux divers services qu'il est possible de mettre en place et aux tâches automatisables. Aujourd'hui, des solution IoT prometteuses commencent à voir le jour, supportées par des moyens d'interactions ergonomiques, comme les enceintes connectées, par exemple, qui permettent d'utiliser des services informatiques par la voix. Toutefois, nous n'en sommes pas encore au point d'oublier totalement la présence des objets connectés, qui nous entourent et beaucoup de solutions attendent toujours une commande directe de la part des utilisateurs : contrôle de l'allumage ou l'extinction de lampes à distance depuis une application mobile par exemple.

Bien sûr, une telle approche pose également de nombreux défis. Il faut être capable de gérer l'**hétérogénéité** des divers objets connectés qui vont participer à la solution pervasive globale. Dans notre exemple précédent, la climatisation peut avoir une interface de contrôle différente de la chaudière et les capteurs de température déployés peuvent avoir chacun leur manière de transmettre leurs données, avec un format ou un protocole de communication particulier. Une solution pervasive doit aussi pouvoir s'adapter au **dynamisme** de l'environnement, avec des objets connectés qui peuvent être ajoutés à tout moment,

d'autres qui peuvent tomber en panne ou bien d'autres encore qui peuvent être mobiles et donc disponibles de manière ponctuelle. Pour revenir à notre exemple, un utilisateur devrait pouvoir librement ajouter ou changer des capteurs de température, voire de nouveaux équipements comme un purificateur d'air. Enfin, un enjeu majeur du domaine est la prise en compte de la **sécurité** des solutions pervasives, en s'assurant, notamment, de l'authenticité des objets ainsi que de la confidentialité et de l'intégrité des données échangées. Diverses attaques, plus ou moins spectaculaires, ont déjà été menées contre des solutions IoT. L'un des cas les plus connus, le *botnet* Mirai, en 2016 [Ant+17], a exploité la faible sécurité et la surexposition de plusieurs milliers d'objets connectés pour les détourner de leur usage d'origine et lancer des attaques par déni de service historiques contre des fournisseurs de services en ligne. Des chercheurs ont également montré qu'il était possible de détourner des objets connectés pour collecter puis exfiltrer des données privées [RS16]. Ces dernières années, les attaques ce sont multipliées avec le déploiement de toujours plus d'objets connectés [Sar+22]. Cela rend d'autant plus nécessaire la recherche, le développement et la mise en place de solutions de sécurité adéquates pour aboutir à des solutions pervasives sécurisées et de confiance. Ma thèse s'inscrit dans cette dynamique, avec le soutien et le financement de la Chaire Trust de la Fondation Grenoble INP, une Chaire industrielle qui a pour thématiques la confiance et la sécurité des systèmes complexes.

La sécurité des systèmes d'information, de manière générale, est un sujet connu et de nombreuses solutions sont disponibles pour répondre à ce besoin. Toutefois, certaines particularités du domaine pervasif peuvent rendre complexes voire inadaptées l'utilisation des solutions de sécurisation les plus classiques.

1.2 PROBLÉMATIQUE

L'utilisation massive d'objets connectés envisagée par l'informatique pervasive demande de considérer la sécurité à une nouvelle échelle. Fondamentalement, les caractéristiques de sécurité qui doivent être assurées dans les systèmes informatiques pervasifs ne diffèrent pas de celles de systèmes informatiques classiques : la confidentialité des données, leur intégrité, l'authentification des équipements et le contrôle de l'accès à différentes fonctionnalités ou données, par exemple. Mais, l'informatique pervasive possède plusieurs particularités qui complexifient le déploiement de solutions de sécurité dites « classiques » :

— **la quantité d'équipements**. Les solutions pervasives demandent la mise en place de larges parcs d'objets connectés et d'infrastructures réseau associées pour dialoguer avec les applications pervasives. Les solutions de sécurité choisies doivent supporter ce passage à l'échelle par rapport aux solutions IoT existantes.
— **l'hétérogénéité des équipements**. Les objets connectés ont des caractéristiques très variés, notamment, en termes de capacité mémoire, de puissance de calcul et d'autonomie énergétique. Selon les contraintes, plus ou moins de ressources seront disponibles pour supporter le fonctionnement d'une solution de sécurité.

— **le dynamisme du parc d'équipements**. Les solutions pervasives doivent accepter facilement de nouveaux objets connectés pour étendre leurs fonctionnalités et assurer leur évolutivité. De plus, certains objets connectés sont mobiles et peuvent intégrer puis quitter des solutions pervasives au gré de leurs déplacements. Les solutions de sécurité auront à gérer ce dynamisme de manière aussi autonome que possible, pour ne pas nuire au principe de transparence de l'informatique pervasive.
— **l'exposition aux menaces**. Les objets connectés des solutions pervasives ont certes pour but d'être intégrés dans l'environnement de façon aussi discrète que possible, il n'est pas garanti qu'ils soient totalement indétectables. Certains objets peuvent être facilement accessibles physiquement, ce qui peut permettre la mise en place de nouveaux vecteurs d'attaques auxquels devront résister les solutions de sécurité.

Face à ces éléments, les solutions classiques utilisées pour la sécurisation des équipements informatiques peuvent se révéler inadaptées. Prenons le cas de l'authentification des équipements. Une solution très répandue est l'utilisation de cryptographie asymétrique avec une ou plusieurs Infrastructures à Clés Publiques (ICP) pour lier une identité à une clé privée, pour diffuser cette information de manière standardisée avec des formats de données comme le certificat X.509 [Coo+08] et pour pouvoir vérifier cette information avec la définition d'Autorités de Certification et des mécanismes de chaînes de confiance. Cette solution est, notamment, utilisée et fonctionnelle dans le cadre de la navigation Web sur l'Internet. Elle permet à des équipements finaux peu contraints en ressources, comme des ordinateurs personnels et des smartphones, d'authentifier des serveurs Web, hébergés dans des locaux dédiés et sécurisés, pour permettre aux utilisateurs une navigation en toute confiance. Dans la navigation Web, le plus couramment, seul le serveur Web est authentifié, tandis que les équipements finaux peuvent accéder au serveur de manière anonyme. Ce faisant, seuls les serveurs ont à gérer l'installation d'un certificat et les opérations associées comme son renouvellement périodique, ce qui limite la charge sur les ICP. Les clients, quant à eux, possèdent une liste de certificats racines, appartenant aux Autorités de Certification, qui leur permet de valider les certificats des serveurs. Cette liste est mise à jour régulièrement et diffusée, notamment, par l'intermédiaire des logiciels de navigation Web.

Si nous cherchons maintenant à appliquer cette solution pour authentifier des objets connectés dans des environnements pervasifs, plusieurs problèmes apparaissent :

— **la complexité des processus**. Dans une solution pervasive, il est raisonnable de vouloir authentifier les serveurs hébergeant les applications pervasives, mais également les objets connectés. Il faut alors, pour chaque objet, générer une clé privée de manière sécurisée, créer et faire signer le certificat associé à cette clé par une Autorité de confiance puis injecter cette clé dans l'objet. Selon le nombre d'objets connectés utilisés, ce processus peut devenir complexe et doit pouvoir passer à l'échelle, en termes de charge sur les infrastructures par exemple. Il faut également considérer le suivi du cycle de vie des clés, ainsi que les opérations de renouvellement du certificat de l'objet et de sa liste de certificats racine. Des opérations dont la complexité augmente également avec le nombre

d'objets et qui peuvent nécessiter des procédures particulières, une mise à jour du logiciel embarqué par exemple.

— **la difficulté de déploiement**. L'authentification par certificat utilise des algorithmes complexes qui demandent des ressources en conséquence pour s'exécuter fluidement. Certains objets connectés, notamment, les plus limités en ressources matérielles embarquées, peuvent avoir des difficultés à exécuter ces algorithmes ou peuvent être impactés négativement par leur exécution, en termes de temps de réponse ou de consommation énergétique, par exemple. Il existe également des protocoles IoT qui ne supportent tout simplement pas l'authentification par certificat, au profit d'autres moyens d'authentification comme l'utilisation d'un secret partagé.

— **le stockage du secret**. Le stockage sécurisé d'un secret long terme comme une clé privée n'est pas une tâche triviale. Encore plus sur des objets connectés qui, contrairement à des serveurs, peuvent être déployés dans des environnements peu maîtrisés et non sécurisés, car les utilisateurs de solutions pervasives ne sont pas des administrateurs réseaux, ni des experts en sécurité informatique. Les objets peuvent alors se trouver exposés à des attaques physiques intrusives. Il existe des mémoires sécurisées, conçus pour résister à ce type d'attaques, mais leur installation engendre des coûts supplémentaires par rapport à l'utilisation de mémoires présentes nativement dans les objets connectés, qui s'expliquent par l'achat de matériel supplémentaire et l'expertise requise pour utiliser efficacement ces mémoires.

— **la gestion des coûts**. L'authentification par certificat a un coût financier non négligeable de base, qui peut comprendre l'achat de certificats à des entreprises spécialisées, le coût de maintenance d'infrastructures locales nécessaires au fonctionnement du processus d'authentification, ou le coût de sécurisation du stockage des secrets comme abordé précédemment. Le coût énergétique, liée à la complexité des algorithmes employés, est également à prendre en compte, en particulier sur des objets connectés conçus pour fonctionner sur batterie avec des objectifs de faible consommation énergétique. Il est probable que le passage à l'échelle du nombre de certificats et des infrastructures associées entraînent une augmentation de ces coûts en fonction du nombre d'objets connectés à gérer.

Ainsi, l'utilisation de certificats et des ICP ne nous paraît pas déployable telle quelle pour assurer, de manière générale, l'authentification entre les objets connectés et les serveurs hébergeant les applications pervasives, voire les applications pervasives elles-mêmes. Ce moyen d'authentification sera probablement utilisé de manière spécifique, dans des solutions pervasives haut de gamme ayant des besoins de sécurité importants, des applications de santé connectée par exemple, et avec des objets possédant de caractéristiques matérielles et logicielles adaptées. Pour des cas d'usage moins critiques, il nous semble nécessaire de réfléchir à la conception de nouvelles solutions de sécurisation adaptées aux particularités des solutions pervasives en termes de passage à l'échelle, de transparence et d'exposition aux attaques physiques, ainsi qu'aux contraintes matérielles et budgétaires des objets connectés utilisés dans ces solutions.

Toutefois, du fait de l'hétérogénéité du domaine pervasif et des objets connectés en particulier, il est peu probable qu'une solution unique puisse sécuriser

de manière satisfaisante l'ensemble des solutions pervasives, et nous pensons plutôt que les objets connectés utiliseront différentes solutions de sécurité selon leurs capacités, comme c'est déjà le cas actuellement pour des solutions IoT. Il faudra alors supporter ces diverses solutions de sécurité pour être compatible avec un large ensemble d'objets connectés, dans le but de restreindre le moins possible le choix des utilisateurs. Cela va complexifier le développement des solutions pervasives, en particulier, celui des applications pervasives qui utiliseront ces objets connectés.

Dans ce contexte où se formeront des ensembles dynamiques d'objets connectés hétérogènes autant d'un point de vue fonctionnel que d'un point de sécurité, **comment-pouvons garantir aux applications pervasives une gestion maîtrisée de leur niveau global de sécurité ?**

1.3 OBJECTIFS

L'objectif principal de ma thèse est de simplifier le développement des applications pervasives en permettant une utilisation à la fois simple et sécurisée des objets connectés de l'environnement, en prenant en compte les besoins de passage à l'échelle, de gestion de l'hétérogénéité et de maîtrise des coûts. Nous souhaitons masquer l'hétérogénéité des objets connectés aux applications pervasives pour qu'elles puissent être focalisées l'utilisation des objets en toute confiance pour rendre des services aux utilisateurs. De plus, nous souhaitons que les utilisateurs puissent déployer de manière simple leurs nouveaux objets connectés et que ces derniers soient sécurisés par défaut. Nous voulons contribuer à l'élaboration de solutions pervasives sécurisées et de confiance pour les utilisateurs.

Nous avons conscience que toutes les technologies n'offrent pas le même niveau de sécurité, c'est pourquoi nous proposons une nouvelle approche pour la modélisation de la sécurité des objets connectés qui se veut transversale, en prenant en compte à la fois la sécurité protocolaire et la sécurité matérielle. Un des objectifs de cette modélisation est de fournir aux applications pervasives une information fiable et accessible sur le niveau de sécurité d'un objet, pour leur permettre d'adapter leur confiance selon le niveau de sécurité des objets utilisés.

Pour déployer ce modèle, nous proposons une approche basée sur une plateforme pervasive faisant l'intermédiaire entre les applications pervasives et les objets connectés. Nous nous sommes basés sur une architecture classique s'appuyant sur une plateforme modulaire éprouvée pour gérer les fonctionnalités de base du domaine pervasif. Partant de cette base, nous proposons une extension dédiée à la gestion de la sécurité qui utilise notre modèle. Notre approche se veut compatible avec les technologies de sécurité connues telles que les certificats X.509 pour assurer la sécurité des communications entre les objets et la plateforme, quel que soit le niveau de confiance accordée aux objets, mais aussi ouverte aux technologies en cours de développements comme l'authentification légère basée sur des éléments matériels.

1.4 ORGANISATION DU DOCUMENT

Après cette introduction, la suite du manuscrit est structurée en deux parties : un état de l'art puis les contributions de la thèse. L'état de l'art comprend deux chapitres :

— le **chapitre 2** présente l'informatique pervasive, en commençant par une définition de cette notion, puis en examinant les éléments clés d'une solution pervasive : des objets connectés, aux plateformes pervasives dans lesquelles sont déployées les applications, en étudiant également les infrastructures réseau de ces architectures. Le chapitre se termine sur la caractérisation des besoins de sécurité que nous avons étudiés pour sécuriser les communications entre les objets connectés et les applications.

— le **chapitre 3** examine de manière concrète la sécurité implémentée dans les éléments présentés au chapitre 1 : les plateformes pervasives, les protocoles de communication réseau et les dispositifs de stockage des secrets présent sur les objets connectés. Nous mettons en lumière la diversité des configurations de sécurité existantes sur ces éléments et les difficultés à prendre en compte cette hétérogénéité. Le chapitre se conclut sur la présentation d'une brique matérielle de sécurité efficace et ayant un faible coût : les fonctions physiques non clonables, en anglais *Physical Unclonable Functions* (PUF). Cette nouvelle technologie n'est pas encore aujourd'hui utilisée classiquement pour sécuriser les objets connectés mais semble, d'après ses caractéristiques, prometteuse pour proposer une solution innovante en termes de sécurité et utilisable pour les applications pervasives.

Les contributions de la thèse sont réparties en deux chapitres :

— le **chapitre 4** détaille le contexte de l'informatique pervasive et la problématique du développement d'applications pervasives sécurisées dans un environnement hétérogène, puis présente notre approche globale basée sur une plateforme pervasive facilitant l'utilisation sécurisée d'objets connectés hétérogènes aux applications pervasives. Cette approche comprend deux contributions principales : une extension pour une plateforme pervasive dédiée à la gestion de la sécurité et un protocole léger utilisant la technologie PUF pour sécuriser à bas coûts les communications avec les objets connectés contraints. Nous présentons l'architecture de ces deux contributions et leurs caractéristiques principales. Nous terminons en montrant comment nos contributions peuvent fonctionner en synergie pour sécuriser de manière transversale et abordable une solution pervasive.

— le **chapitre 5** présente le développement de démonstrateurs pour valider nos travaux : une base de plateforme pervasive modulaire et sécurisée implémentant notre extension de sécurité et les implémentations de notre protocole de sécurisation PUF sur une cible embarquée et au sein d'une librairie de communication sécurisée. Le démonstrateur de la plateforme nous a permis d'évaluer les gains offerts par notre extension en termes d'aisance dans le développement d'applications et l'utilisation sécurisée d'objets connectés hétérogènes. Les implémentations de notre protocole nous ont permis de collecter des métriques que nous avons utilisées pour

comparer notre contribution avec des travaux similaires et conclure sur notre apport dans la sécurisation des objets connectés contraints.

Enfin, le **chapitre 6** synthétise les points clés de nos travaux en rappelant nos contributions pour faciliter le développement d'applications pervasives sécurisées. Nous présenterons également, dans ce chapitre, les perspectives envisagées pour de futurs travaux.

Première partie

APPLICATIONS PERVASIVES ET SÉCURITÉ

2 INFORMATIQUE PERVASIVE

Nous présentons dans ce chapitre le concept d'informatique pervasive, une vision de l'informatique très intégrée à l'environnement et qui a pour but de rendre service aux utilisateurs sans qu'ils en aient pleine conscience. Nous allons voir les bénéfices qui peuvent être apportés par ce concept, mais également les défis autour de sa mise en œuvre, notamment sur le volet de la sécurité du système qui est un sujet à la fois crucial et complexe.

Nous commencerons par définir ce concept et nous en présenterons les principaux éléments : les objets connectés, puis le réseau et, enfin, les applications pervasives. Nous finirons le chapitre en définissant les principaux besoins de l'informatique pervasive en termes de caractéristiques de sécurité, et en illustrant ces besoins dans des situations concrètes.

2.1 DÉFINITION

L'informatique pervasive puise sa source dans la vision de Mark Weiser, alors docteur de l'Université du Maryland et directeur du Xerox Palo Alto Research Center. Dans un article de 1991 [Wei91], M. Weiser présente le concept d'*ubiquitous computing* et ses principales caractéristiques : invisibilité et amélioration du monde existant. Quelques années plus tard, le terme *pervasive computing* est également utilisé [AS99 ; Mar99] pour désigner un concept similaire qui vise une intégration poussée des ordinateurs dans l'environnement, notamment par l'entreprise IBM qui est active sur le sujet [SM03]. Dans le cadre de nos travaux sur la sécurité des objets connectés, nous n'avons trouvé de différences majeures entre ces deux concepts. Dans la suite de ce document, nous assumerons que ces concepts sont équivalents dans leurs grands principes et nous utiliserons le terme unique d'informatique pervasive. Ainsi, il s'agit de rendre l'informatique omniprésente ; mais invisible aux yeux des utilisateurs. Invisible dans le sens où les utilisateurs seraient amenés à interagir et à bénéficier de l'informatique pervasive sans le réaliser, de manière naturelle.

Pour mettre en place cette vision de façon concrète, M. Weiser identifie trois briques technologiques :

> "The technology required for ubiquitous computing comes in three parts : cheap, low-power computers that include equally convenient displays, software for ubiquitous applications and a network that ties them all together." [Wei91]

Nous pouvons représenter schématiquement un système informatique pervasif par la figure 2.1, qui fait apparaître les trois éléments principaux identifiés précédemment :

— les objets connectés, qui sont des équipements intégrés à l'environnement de l'utilisateur,
— les applications pervasives, qui représentent la partie logicielle du système, et
— l'infrastructure réseau, qui fait la liaison entre les deux éléments précédents.

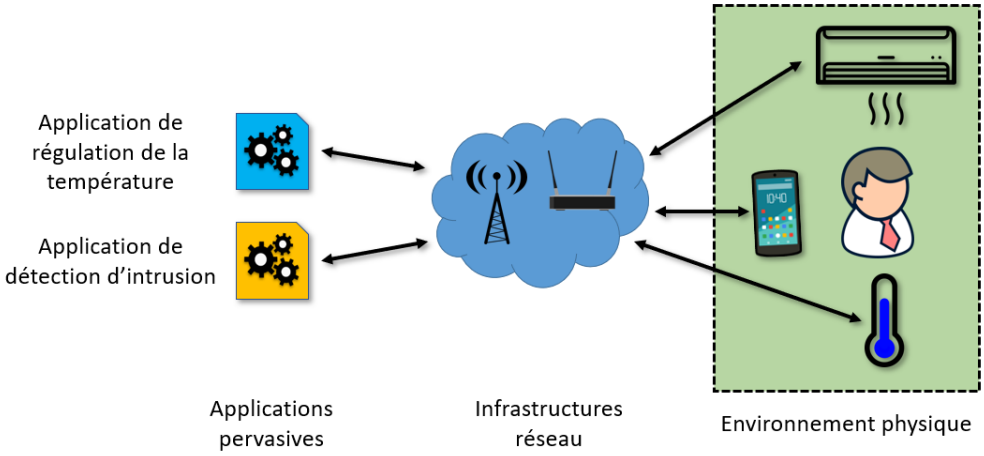

FIGURE 2.1 – Vue d'ensemble d'un système informatique pervasif.

Un dernier élément fondamental en informatique pervasive est la notion de contexte. Par contexte, nous considérons la définition donnée par A. Dey en 2001 [Dey01] :

> "Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves."

Des informations telles que la température d'une pièce, le nombre de personnes se trouvant dans une maison ou le rythme cardiaque d'une personne peuvent être des informations de contexte utiles au fonctionnement d'une application pervasive.

Illustrons concrètement à quoi peut ressembler un système informatique pervasif à travers deux exemples : un système de régulation de la température et un système d'alarme.

Le système de régulation pervasif a pour objectif d'adapter en temps réel la température d'une habitation pour atteindre une température consigne. Dans un cadre pervasif, cette consigne n'est pas directement communiquée par un utilisateur mais inférée à partir d'un contexte [Ger+17] : heure de la journée, météo extérieure, présence ou non d'usagers dans l'habitation, etc. participent à l'élaboration d'une valeur consigne. Pour mener à bien cet objectif, le système pervasif s'appuie sur divers objets connectés répartis dans l'habitation : capteurs de températures, détecteurs de présence, détecteurs d'ouverture des portes et des fenêtres et climatisation. Les données remontées par ces objets peuvent être

plus ou moins sensibles, du point de vue de l'utilisateur et/ou du point de vue de l'application. Si les données d'un détecteur de présence sont publiquement accessibles, cela peut révéler des informations confidentielles sur les habitudes d'un foyer et porter atteinte à la vie privée des utilisateurs. Si un capteur de température remonte des données incorrectes ou dont l'intégrité ne peut être vérifiée, la vision de l'application de régulation sera biaisée et elle risque de mal fonctionner. C'est pourquoi il est important de pouvoir faire confiance aux objets déployés sur le terrain et utilisés par des applications pervasives.

Prenons un exemple plus critique encore avec un système d'alarme pervasif, dont le but est de détecter des intrusions dans l'habitation et d'alerter les usagers à ce sujet ainsi que les services compétents. Un tel système s'appuie sur des objets connectés tels que des caméras, des détecteurs de présence, des détecteurs d'ouverture et des sirènes. A noter que certains de ces capteurs peuvent être partagés avec le système de régulation présenté précédemment, ce qui demande de gérer les éventuels conflits d'accès entre les applications [Had18]. Dans un système sensible comme celui-ci, la confiance dans les objets et les données remontées est d'autant plus importante. En effet, si un détecteur de présence remonte des données erronées, cela pourrait empêcher l'alarme de se déclencher alors qu'un cambriolage est en cours.

Ces deux exemples illustrent l'importance de la prise en compte de la sécurité pour assurer le bon fonctionnement des solutions pervasives. Ces dernières sont particulièrement exposées du fait des nombreux éléments dont elles sont constituées : les objets connectés, les applications pervasives et les réseaux de communications interconnectant le tout. Idéalement, il faudrait non seulement sécuriser chaque élément unitairement, mais également toutes les interactions entre éléments, telle que la remontée de données d'un capteur jusqu'à une application pervasive. De plus, cette sécurisation se faire de manière autonome, sans interaction de la part des utilisateurs pour adhérer à la vision pervasive. C'est un équilibre complexe à atteindre. Nous allons définir chacun des éléments composant une solution pervasive dans les sections suivantes, puis nous terminerons en définissant les besoins de sécurité de ces solutions.

2.2 OBJET CONNECTÉ

Les progrès de l'électronique, notamment en termes de miniaturisation, de consommation énergétique et de puissance de calcul, ont permis la création de circuits et d'ordinateurs de plus en plus réduits. Là où il fallait leur dédier une salle entière à l'époque des premiers ordinateurs, les ordinateurs personnels sont désormais relativement compacts et nous avons même assisté à l'avènement des micro-ordinateurs. Le téléphone intelligent ou *smartphone*, aujourd'hui très répandu, est certainement le représentant le plus connu de ces micro-ordinateurs qui sont très facilement transportables.

De plus, le développement de protocoles de communication de plus en plus performants a permis d'apporter de la connectivité à ces systèmes miniaturisés. Cela a donné naissance à des concepts comme les Systèmes Cyber-Physiques [LK10] ou l'Internet des Objets [Into5]. Dans ces concepts, des systèmes informatiques miniaturisés sont capables de surveiller et contrôler des entités du monde physique. En plus de cela, ces systèmes exposent leurs capaci-

tés à des entités extérieures grâce à leur connectivité réseau. Il faut noter que ces concepts sont extrêmement vastes et peuvent concerner des cas d'usages allant de la télémédecine [RA17 ; LLR13] aux villes connectées [Lee+20] en passant par l'Industrie 4.0 [Yaq+18 ; Lal+21]. Concernant les objets connectés, il n'existe pas aujourd'hui à notre connaissance de définition qui fasse consensus. Par exemple, il existe plusieurs ontologies pour décrire et caractériser des objets connectés [TS19]. Dans cette thèse, nous avons utilisé la définition générale donnée par B. Dorsemaine [Dor+15] :

> "Sensor(s) and/or actuator(s) carrying out a specific function and that are able to communicate with other equipment. It is part of an infrastructure allowing the transport, storage, processing and access to the generated data by users or other systems."

Nous retrouvons dans cette définition les principales caractéristiques d'un objet connecté : la réalisation d'une tâche spécifique à l'aide de capteurs et/ou d'actionneurs embarqués, et la capacité de communication avec d'autres équipements pour pouvoir, notamment, intégrer une infrastructure de gestion des données et de contrôle des objets. Cette définition peut être implémentée de diverses manières et, dans les faits, nous trouvons sur le marché une grande diversité d'objets pouvant y correspondre. Ces objets varient en taille, en ressources embarquées ou en autonomie énergétique. En reprenant le système de régulation de température présenté dans la section 2.1, il est certain qu'il existe des différences notables entre un bloc de climatisation pilotable et un capteur de température sur les caractéristiques citées précédemment. Or, ces différences vont influencer sur les solutions de sécurité qu'il sera possible de déployer sur chaque objet.

Durant cette thèse, nous nous sommes particulièrement intéressés au cas des objets connectés contraints qui disposent de ressources limitées pour effectuer leurs tâches et, notamment, pour implémenter des solutions de sécurité. Ces objets sont principalement destinés à des cas d'usage jugés non critiques, comme des applications domotiques. Toutefois, ils peuvent dans ces situations collecter et échanger des données privées sur les utilisateurs et c'est pourquoi nous pensons que ces objets se doivent de proposer un niveau minimal de sécurité. Le défi étant de trouver des solutions de sécurité adaptées aux contraintes des objets. Pour caractériser ces contraintes, nous nous sommes appuyés sur les travaux réalisés par l'IETF [1] qui ont donné lieu à la RFC 7228 [CMA14]. Ce document définit plusieurs types de contraintes et les niveaux associés :

— une contrainte en termes de mémoire embarquée. Il s'agit de l'espace de stockage dont un objet connecté dispose. Il est communément séparé en deux parties : une partie mémoire morte, qui stocke du contenu persistant, notamment le code embarqué qui implémente les fonctionnalités de l'objet, et une partie mémoire vive, qui est utilisée pour la manipulation de données lors du fonctionnement de l'objet. Les niveaux de contraintes varient de C0 à C2 et sont présentés dans le tableau 2.1.

1. *Internet Engineering Task Force.*

Niveau	Taille de la mémoire morte	Taille de la mémoire vive
C0	Inférieure à 100 Kio [2]	Inférieure à 10 Kio
C1	Environ 100 Kio	Environ 10 Kio
C2	Environ 250 Kio	Environ 50 Kio

TABLE 2.1 – Niveaux associés à la contrainte mémoire définie par la RFC 7228.

Plus ces espaces sont faibles et plus les fonctionnalités d'un objet doivent être implémentées de manière optimisée. L'implémentation de certains mécanismes de sécurité a un coût en espace mémoire, coût qui peut parfois être trop élevé selon les contraintes et l'espace réservé pour les fonctionnalités.

— une contrainte relative à la source d'énergie de l'objet. Il s'agit notamment d'examiner le dimensionnement de la source d'énergie et les modes de recharge de cette source s'ils existent. Quatre niveaux sont définis pour cette contrainte : E0, E1, E2 et E9. Ils sont présentés dans le tableau 2.2.

Niveau	Type de limitation énergétique	Exemple
E0	Evénementielle	Alimentation sur demande
E1	Périodique	Batterie rechargeable
E2	Durée de vie	Batterie non rechargeable
E9	Pas de limitation	Branchement sur secteur

TABLE 2.2 – Niveaux associés à la contrainte énergétique définie par la RFC 7228.

Dans le cas des objets fonctionnant sur batterie, en particulier ceux sur batterie non rechargeable, le coût énergétique des fonctionnalités doit être pris en compte pour assurer la durée de vie souhaitée. L'ajout de mesures de sécurité peut avoir un coût énergétique non négligeable, du fait des opérations supplémentaires requises.

Suivant cette terminologie, nous utiliserons le terme d'objet connecté contraint pour un objet ayant une contrainte mémoire de niveau C1 et une contrainte énergétique de niveau E1 ou E2. Cela peut correspondre à des capteurs ou des actionneurs domotiques nomades, comme par exemple une sonde de température autonome ou un capteur de présence fonctionnant sur batterie. D'autres contraintes peuvent s'appliquer à un objet connecté, telles que les contraintes réseau [Vas14] qui peuvent limiter la quantité de données échangées ou la disponibilité du médium de communication. Nous nous sommes cependant concentrés sur les contraintes d'espace mémoire et d'énergie car nous jugeons que ce sont les principales contraintes qui impactent le déploiement de mécanismes de sécurité sur les objets connectés. Ainsi, il est important de disposer de solutions pouvant apporter un niveau de sécurité correct tout en étant légères en termes d'empreinte mémoire et de calculs complexes. Bien que les objets connectés contraints et *low-cost* ne soient pas destinés à des cas d'usage sensibles dans des domaines comme la défense ou la santé connectée, leur sécurisation est importante pour éviter des intrusions et des vols de données à grande échelle.

Ces objets doivent disposer d'un niveau de sécurité basique pour permettre leur intégration dans des solutions pervasives en toute confiance.

2.3 APPLICATION PERVASIVE

Une application est un logiciel qui effectue des tâches spécifiques pour des utilisateurs. Comme tout logiciel, elle doit être déployée et s'exécuter sur une machine qui peut prendre de nombreuses formes, allant de l'ordinateur de bureau à l'objet connecté. Cependant une application est rarement déployée directement sur une machine. Elle est souvent déployée sur un système d'exploitation, un logiciel particulier qui gère les aspects dit bas-niveau des machines, comme l'initialisation du matériel au démarrage. Ce logiciel a l'avantage de mettre à disposition d'autres logiciels, telles que les applications, des interfaces facilitant l'usage des ressources et des fonctionnalités de la machine sous-jacente. Cela a pour effet de faciliter le développement des applications. Il est également possible de déployer une application sur un intergiciel (ou *middleware*), un autre type de logiciel qui vient se placer entre le système d'exploitation et l'application. Un intergiciel a généralement pour but de fournir des services de plus haut niveau aux applications, en prenant à sa charge l'utilisation des interfaces du système d'exploitation. Dans ce cas, l'application sera déployée sur l'intergiciel et utilisera les interfaces que ce dernier met à disposition. Comme pour le système d'exploitation, l'utilisation d'un intergiciel a pour objectif de faciliter le développement des applications. Une application pervasive est un type particulier d'application, qui sera donc déployée sur un système d'exploitation ou un intergiciel dédié. Ce type d'application possède des propriétés clés liées à la vision pervasive de l'informatique. Ces propriétés sont les suivantes [Gün14] :

— **Gestion dynamique de ressources**. Une application classique est conçue pour fonctionner avec un panel de ressources propres à sa machine hôte (temps CPU, mémoire vive, stockage), dites ressources virtuelles et relativement statiques (on peut réaliser en avance une estimation des besoins vis-à-vis de ces ressources). Dans le cas d'une application pervasive, en plus de cette dimension de gestion des ressources de la machine hôte, l'application fait usage de ressources distantes. Elles prennent notamment la forme d'objets connectés tels que présentés dans la Section 1.2. Ces ressources distantes sont plus délicates à gérer que les ressources locales. Elles peuvent nécessiter des protocoles d'accès particuliers ou être partagées entre plusieurs applications [Had18 ; Rot+18]. Un intergiciel peut être en charge de la découverte et de la gestion de certaines ressources distantes, les objets connectés par exemple [Mau+12].

— **Prise en compte du contexte**. Nous l'avons vu dans la définition de l'informatique pervasive, la notion de contexte est primordiale [Dey01]. La construction d'un contexte est cependant une tâche difficile, qui nécessite de croiser de nombreux flux de données [Ayg17] (température, heure de la journée, présence d'un utilisateur). De plus, les informations de contexte pertinentes varient d'une application à une autre selon la finalité attendue. Une application de détection d'intrusion n'aura pas les mêmes besoins de contexte qu'une application de gestion de la température. Cela dit, les flux de données utilisés comme base pour la construction de différents

contextes peuvent être mutualisés par un intergiciel et mis à disposition au travers de services de contexte dédiés pour faciliter leur accès.

— **Adaptabilité**. Une application classique est généralement conçue pour exécuter un nombre limité de scénarios dans environnement informatique relativement statique et connu d'avance, voire contrôlé : un serveur par exemple. Une application pervasive, en revanche, doit s'adapter à l'environnement de chaque utilisateur, qui est libre d'y installer les objets connectés qu'il souhaite pour répondre à des besoins spécifiques. De plus, ce type d'environnement évolue dynamiquement au gré, notamment, de l'ajout ou du retrait d'objets. L'application pervasive doit pouvoir réagir à ces évolutions sans intervention d'un opérateur humain. Ainsi, les applications pervasives doivent être conçues de sorte à pouvoir s'adapter, lors de leur exécution et de façon autonome, à des changements dans leur environnement [Ger+17].

— **Sécurité**. La sécurité des applications pervasive est primordiale. Il est nécessaire de sécuriser l'accès aux ressources en veillant à ce qu'elles soient authentiques et de confiance. Il peut également être souhaitable de mettre en place des mécanismes de contrôle d'accès pour limiter l'usage de certaines ressources ou de certaines fonctionnalités selon le rôle de l'utilisateur qui effectue la demande ou parfois selon des contraintes temporelles [Had18] : par exemple, restreindre l'accès aux interfaces de configuration des capteurs aux applications dédiées à l'administration du parc de capteurs. De plus, les données manipulées par les applications pervasives peuvent exposer la vie privée des utilisateurs [Far13]. C'est le cas des données de contexte qui, selon l'application, peuvent chercher à modéliser des situations privées comme la présence ou non d'une personne à son domicile, ou bien encore l'état de santé d'un patient. Une plateforme pervasive peut offrir des fonctionnalités comme la sécurisation des communications pour faciliter le respect des exigences de sécurité. Nous présenterons en détails les besoins de sécurité des applications pervasives dans la section 2.5.

Dans nos travaux, nous avons fait le choix de nous concentrer sur les plateformes comme support des applications pervasives, bien que ces dernières ne soient en rien obligatoires pour la mise en place de solutions pervasives. L'utilisation d'une plateforme permet la séparation des préoccupations : les développeurs d'applications peuvent se concentrer sur leur partie fonctionnelle tandis que les développeurs de la plateforme gèrent les besoins transversaux tels que l'hétérogénéité, le partage des ressources ou la sécurité [QM10]. La figure 2.2 illustre le déploiement d'une solution pervasive avec une plateforme.

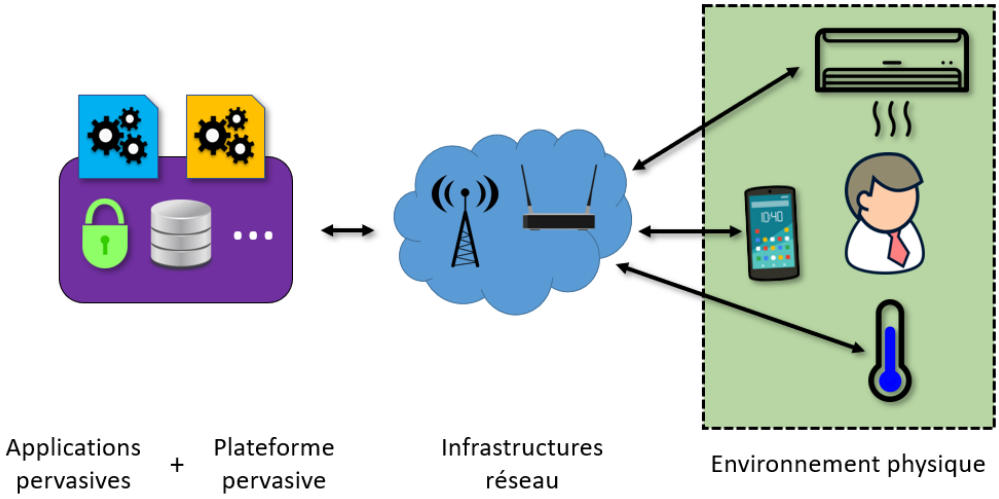

FIGURE 2.2 – Système informatique pervasif avec plateforme pervasive.

Les objets connectés sont déployés dans l'environnement physique où ils peuvent rendre des services à l'utilisateur. Ils communiquent uniquement avec la plateforme pervasive via diverses infrastructures réseaux. Nous présenterons plus précisément les architectures réseaux possibles dans la section 2.4. La plateforme pervasive gère la connexion des objets et elle permet le déploiement des applications pervasives. Elle peut fournir aux applications divers services de support, tels que la persistance de données [ECL14] ou la découverte de services et d'objets connectés [Mau+12]. Certaines plateformes peuvent également s'interconnecter avec d'autres plateformes compatibles pour fournir des services supplémentaires [Bec+04]. Nous pensons que, du fait de son rôle d'intermédiaire entre les applications pervasives et les objets connectés, la plateforme pervasive est une position intéressante pour déployer des outils et des services de sécurisation, comme la gestion des échanges avec les objets connectés de manière sécurisée par exemple.

Il faut toutefois noter que l'utilisation d'une plateforme pervasive n'est pas sans contraintes. En particulier, cela engendre un certain niveau d'abstraction pour les applications pervasives par rapport à une utilisation directe des objets connectés. En effet, ces dernières ont accès seulement aux interfaces de communication exposées par la plateforme, puis c'est cette dernière qui décide de la manière dont elle communique avec les objets connectés. Ce comportement peut être plus ou moins paramétrable, selon les choix de développement effectués. Cela peut être un problème pour des applications qui nécessitent l'utilisation de fonctionnalités particulières d'un objet ou d'un protocole donné. Il faut trouver un juste équilibre entre le niveau d'abstraction des interfaces et la liberté d'action laissée aux applications.

De nombreuses recherches ont été menées sur le sujet des plateformes pervasives. Diverses architectures ont été proposées [Bec+19] telles que iCasa [Lal+14], ubiREST [Cap+14], LinkSmart [ERA10] ou encore In.IoT [da +21]. Le secteur privé et, notamment, les entreprises ayant déjà des activités dans les réseaux et les services web, s'est également intéressé au sujet. Des plateformes commerciales pour faciliter la conception et le déploiement de solutions exploitant l'Internet des Objets ont vu le jour. Nous pouvons, par exemple, citer Orange

Live Objects [3], Microsoft Azure IoT [4], AWS IoT [5] ou Google Cloud IoT [6]. Nous étudierons certaines de ces solutions et la façon dont elles abordent la sécurisation des objets connectés dans la section 3.1.

2.4 INFRASTRUCTURE RÉSEAU

Après avoir présenté les objets connectés puis les applications et le plateformes pervasives dans les sections précédentes, nous présentons dans cette section le dernier élément clé de l'informatique pervasive : l'infrastructure réseau. Plusieurs architectures réseaux existent pour permettre la communication entre les objets connectés et les applications pervasives [Omo+19], que nous supposons désormais hébergées sur une plateforme pervasive. Nous allons présenter, dans cette section, les deux approches principales qui se différencient par la répartition géographique entre les objets connectés et la plateforme, les ressources disponibles en termes de puissance de calcul et de stockage, le délai d'échange des informations et la sécurité.

2.4.1 *Cloud computing*

Le *cloud computing* [MES08] fait référence au déplacement de la puissance de traitement informatique de machines hébergées localement vers des machines hébergées à distance par des entreprises dédiées. Cela concerne, par exemple, le stockage de données ou l'exécution d'applications. Le modèle *cloud* s'appuie notamment sur de gigantesques fermes de serveurs, appelées *datacenters*, qui sont administrées par des entreprises dédiées, les *clouds providers*. Ce modèle possède de nombreux avantages, parmi lesquels :

— la disponibilité quasi-immédiate de ressources informatiques ;
— de la flexibilité sur les ressources allouées au système, qui peuvent être augmentées ou diminuées dynamiquement, en fonction des besoins ;
— une portée importante permettant la connexion de nombreux objets, grâce à l'utilisation de réseaux étendus (*Wide Area Network* ou *WAN*, en anglais) comme, notamment, l'Internet.

Les applications pervasives basées sur l'Internet des Objets ont de forts besoins en termes d'espace de stockage et de puissance de calcul, pour traiter les importantes masses de données générées par les objets connectés déployés sur le terrain [Pie+20]. Le *cloud* est à même de répondre à ces besoins et les entreprises du domaine, telles que Microsoft, Amazon ou Baidu se sont rapidement adaptées en développant des offres et des services *cloud* dédiés aux cas d'usage IoT. La figure 2.3 illustre la mise en réseau d'objets avec le modèle *cloud*.

3. https ://liveobjects.orange-business.com/#/liveobjects
4. https ://azure.microsoft.com/fr-fr/solutions/iot/
5. https ://aws.amazon.com/fr/iot/
6. https ://cloud.google.com/iot-core

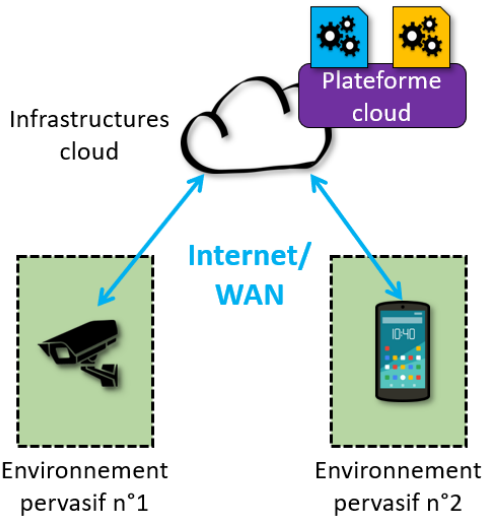

FIGURE 2.3 – Approche cloud pour l'interconnexion d'objets.

Dans ce modèle une plateforme pervasive, telle que présentée dans la section 2.3, est déployée sur les infrastructures *cloud* d'un fournisseur et accueille les différentes applications pervasives. Les objets connectés sont, eux, déployés dans leurs environnements. Ils communiquent avec la plateforme via des réseaux étendus tels qu'Internet. Les objets et les environnements supervisés ne sont pas nécessairement à proximité les uns des autres. Dans notre schéma par exemple, les environnements pervasifs n°1 et n°2 peuvent être très éloignés géographiquement, pourvu que chaque environnement dispose d'un accès réseau adéquat. La plateforme *cloud* centralise la remontée des données de tous les objets et elle permet aux applications d'accéder à ces données. La communication bidirectionnelle est souvent supportée pour permettre également l'envoi de données aux objets, des commandes ou une nouvelle configuration, par exemple.

Bien que le modèle *cloud* présente de nombreux avantages, comme ceux cités en début de section, des études ont également mis en avant certaines limitations de ce modèle dans des scénarios pervasifs et IoT [Omo+19 ; Pie+20 ; Lal23] :

— un passage à l'échelle du modèle limité, à cause, notamment, de l'augmentation massive des données générées par les objets et les risques de congestion des réseaux associés ;
— une latence élevée, du fait de la distance entre les objets connectés et les *datacenters* ;
— des coûts de communication, matériels et financiers, élevés du fait de l'envoi régulier de données vers les *datacenters* via des réseaux étendus ;
— des risques de sécurité accrus, avec la centralisation des données collectées et la connexion des objets à des réseaux publics qui les exposent à des acteurs malveillants.

Ces limitations rendent délicates le développement de certaines applications pervasives avec le modèle *cloud*, en particulier des applications interactives ou immersives, qui nécessitent de la réactivité tout en traitant une quantité importante de données. Du fait de ces limitations, d'autres modèles ont été envisagés pour, notamment, réduire la latence des échanges de données.

2.4.2 *Edge computing*

Dans le contexte IoT, les données sont produites en majorité par des objets placés dans des environnements physiques, en bordure des réseaux de communication. Plutôt que de faire transiter ces données vers des infrastructures de traitement situées au cœur des réseaux comme c'est le cas de l'approche *cloud computing*, il est possible de placer en bordure de réseau les processus de traitement des données : c'est l'approche *edge computing* [Shi+16]. Il existe trois implémentations principales de l'approche *edge* [DD17] : le *Mobile Edge computing* [Bec+14], le *cloudlet computing* [Sat+09] et le *fog computing* [Bon+12]. Le *Mobile Edge Computing* consiste à déployer des capacités de traitement au niveau des stations de base des réseaux mobiles, elle ne traite pas les autres types de réseaux. Le *cloudlet* s'appuie sur l'installation de datacenters de taille réduite à proximité des environnements pervasifs. Enfin, le *fog computing* fait usage d'équipements situés dans l'environnement même des utilisateurs et capable d'effectuer des traitements de données, un routeur Wi-Fi par exemple. Durant ma thèse, nous avons choisi de nous concentrer sur le *fog computing* car il s'agit pour nous de l'implémentation la plus pertinente pour les cas d'usages domotiques sur lesquels nous nous sommes concentrés, du fait du déploiement d'un équipement au sein même de l'environnement utilisateur contrairement aux deux autres implémentations présentées.

Le concept de *fog computing* a été initialement proposé par des chercheurs de Cisco en 2012 [PM18]. L'idée principale de ce modèle est de déployer du traitement de données au niveau du réseau local (*Local Area Network* ou *LAN* en anglais), dans une approche plus distribuée que le modèle *cloud*. Le modèle s'articule autour d'équipements dédiés, souvent désignés par les termes passerelle ou *gateway*, qui sont déployés sur les réseaux locaux. Chaque passerelle assure la gestion des objets locaux : elle récupère les données remontées par les objets, peut effectuer des traitements sur ces données et envoyer des commandes aux objets. La figure 2.4 illustre la mise en réseau d'objets avec le modèle *fog*.

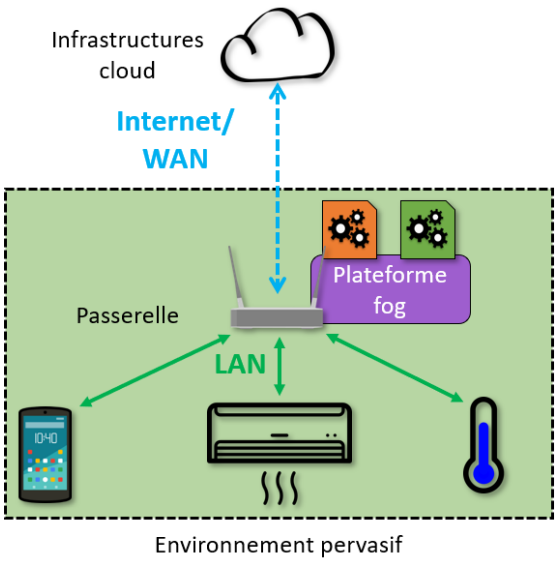

FIGURE 2.4 – Approche fog pour l'interconnexion d'objets.

Ici, les objets sont connectés à un réseau local sur lequel est également présente la passerelle. Les objets utilisent ce réseau pour communiquer avec la passerelle, pour remonter de la donnée ou recevoir une commande par exemple. L'utilisation d'un réseau local limite la dispersion spatiale des objets. Dans le modèle *fog computing*, une passerelle couvre généralement les objets d'une maison ou d'un immeuble par exemple.

Une passerelle possède moins de ressources que le *cloud*, c'est pourquoi les modèles *fog* et *cloud* ne sont pas opposés mais complémentaires. La passerelle traite les demandes locales les plus simples et peut transférer à des services *cloud* les demandes dont le traitement nécessite plus de ressources [Bon+14]. Le modèle *fog* permet de diminuer la latence de communication en se rapprochant de l'environnement de déploiement des objets. La congestion des réseaux est également réduite grâce aux traitement locaux qui sont effectuées par la passerelle, réduisant le nombre de requêtes en direction du *cloud*. Cela rend le modèle très attractif pour le développement de systèmes informatiques pervasifs du fait de la réactivité permise par la faible latence. De plus, la proximité des passerelle avec le milieu utilisateur peut aider à la collecte d'information de contexte [Bon+12], qui sont un élément clé de tout système pervasif.

Le *fog computing* reste un modèle récent avec de nombreux sujets de recherche ouverts [Omo+19], allant de l'interfaçage efficace entre le *fog* et le *cloud* à la gestion de l'hétérogénéité des objets et des protocoles sur les réseaux locaux. Des sujets que nous avons gardé à l'esprit lors de nos travaux.

2.4.3 *Exemple d'architecture réseau globale*

Comme nous l'avons vu dans la section précédente, les différentes approches présentées pour permettre la communication entre des applications et des objets connectés ne sont pas mutuellement exclusives. Au contraire, plusieurs recherches soulignent la complémentarité des approches *cloud* et *fog* [Omo+19 ; Bon+14]. La figure 2.5 illustre une architecture réseau globale pour une solution connectée qui intègre les approches présentées précédemment.

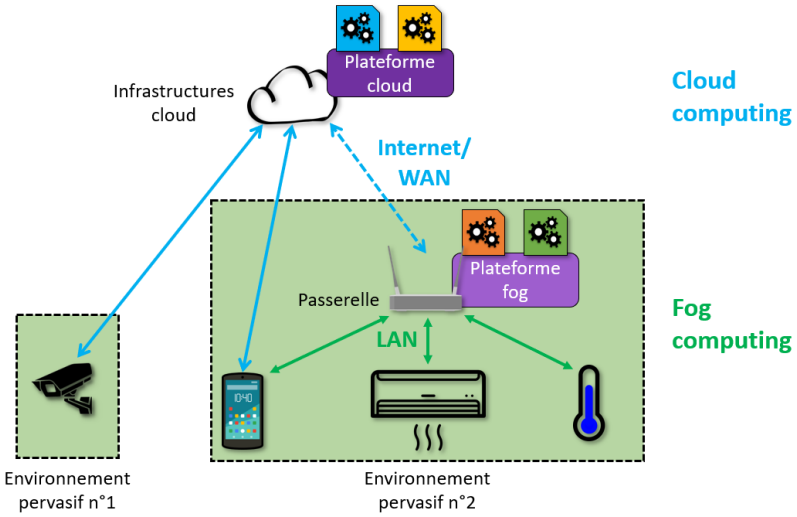

FIGURE 2.5 – Un exemple d'architecture réseau globale pour un système informatique pervasif.

Dans cet exemple, une partie des objets est connectée directement à une plateforme *cloud*. Il peut s'agir d'objets générant de grandes quantités de données, nécessitant de surcroît des traitements supplémentaires avant d'être exploitable par une application. Les ressources disponibles en abondance dans l'environnement *cloud* peuvent alors être mobilisées pour effectuer ces traitements. Une autre partie des objets est connectée à une plateforme *fog*, installée sur une passerelle locale. Les applications déployées sur la plateforme *fog* disposent d'informations plus localisées du fait de la connexion avec des objets de l'environnement proche uniquement. Ces applications peuvent également être plus réactives aux changements de l'environnement local et réagir en conséquence via la commande d'actionneurs, grâce à la faible latence des liens locaux. Les ressources de la passerelle permettent d'effectuer la plupart des traitements nécessaire à la gestion du parc local. Toutefois, la passerelle peut faire ponctuellement appel à des services hébergés dans le *cloud*, pour récupérer des données complémentaires ou effectuer des traitements complexes comme de la reconnaissance d'image, par exemple.

2.5 SÉCURITÉ

Si l'informatique pervasive promet des environnements plus intelligents et plus ergonomiques, l'intégration massive d'équipements informatiques dans les lieux publics et privés n'est pas sans risques. Dans son article de 1991, M. Weiser imaginait déjà les possibles atteintes à la vie privée des utilisateurs que pouvait engendrer l'informatique pervasive [Wei91]. Ce n'est en fait qu'un des risques potentiels pouvant toucher ces systèmes qui sont par nature très exposés. Par exemple, les objets connectés faisant partie d'une solution pervasive et déployés sur le terrain peuvent se retrouver exposés à des attaques physiques intrusives visant à extraire les secrets qu'ils stockent. L'usage de protocoles de communication sans fil peut permettre des collectes passives de données, d'une manière difficilement détectable. Enfin, le besoin de dynamisme et d'ajout facile de nouveaux objets peut être exploité pour intégrer des objets malicieux au sein des solutions pervasives.

Pour ces raisons, la sécurité des environnements pervasifs est un sujet crucial. Il est possible de prendre appui sur les modèles de sécurité des systèmes informatiques classiques, mais en tenant compte des défis propres au domaine pervasif [HA06] tels que l'hétérogénéité des équipements ou le fonctionnement autonome des solutions. Ainsi, nous avons choisi de concentrer nos recherches sur les caractéristiques de confidentialité et d'intégrité des données [Avi+04], ainsi que sur l'authentification des équipements et les mécanismes de contrôle d'accès liés [HA06]. De notre point de vue, l'examen de ces caractéristiques couvre un large éventail de problématiques de sécurité tels que l'intrusion d'objets adverses dans une solution pervasive ou la récupération de secrets via l'écoute de réseaux. Toutefois, nous n'avons pas traité en profondeur dans cette thèse du sujet de la protection de la vie privée des utilisateurs, dont nous reconnaissons l'importance mais qui sort du périmètre de nos travaux qui sont plus centrés sur les objets connectés. Nous avons également fait le choix délibéré de nous concentrer uniquement sur la sécurité des solutions pervasives, en mettant de côté la sûreté de fonctionnement dont nous reconnaissons l'importance pour aboutir à des solutions de confiance.

Dans le reste de cette section, nous allons définir chacune des caractéristiques de sécurité que nous avons choisi de traiter et concrétiser les risques associés en prenant appui sur des cas d'usage pervasifs concrets.

2.5.1 *Confidentialité*

Pour le concept de **confidentialité**, nous adoptons la définition suivante :

> "[the] absence of unauthorized disclosure of information." [Avi+04]

La confidentialité concerne tout type d'informations ou de données manipulées par un système informatique pervasif : il peut s'agir de données de terrain collectées par des objets connectés, puis traitées par une application, de clés informatiques utilisées pour réaliser du chiffrement ou d'identifiants entrés par un utilisateur, par exemple.

Si la confidentialité n'est pas bien prise en compte, le principal risque de sécurité est le vol de données sensibles [Zha+17]. Deux exemples de ce type de risque sont illustrés par la figure 2.6.

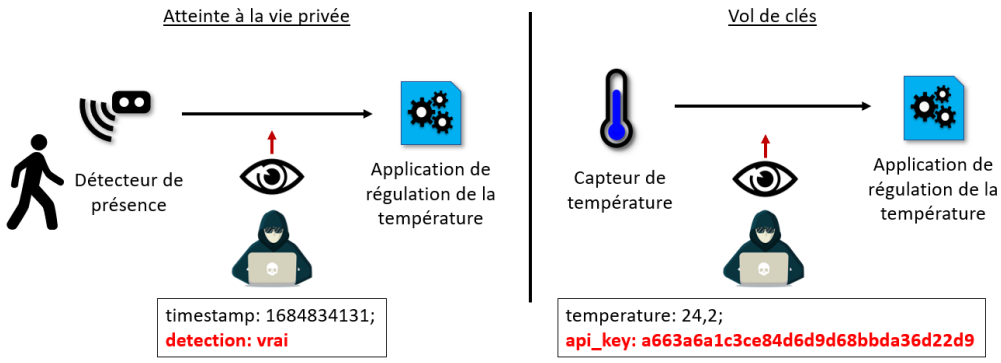

FIGURE 2.6 – Deux exemples de vol de données dans le domaine des objets connectés.

Dans le premier exemple (illustration de gauche), nous prenons le cas d'un détecteur de présence situé dans le logement. Ce capteur informe une application de régulation de la température de la présence d'utilisateurs à un endroit du logement. L'application peut utiliser cette information pour adapter la température des pièces en fonction de leur fréquentation, par exemple. Si cette information est mal protégée, un attaquant pourra y accéder et il sera capable de connaître ou d'estimer la fréquentation du logement ciblé. Il s'agit là d'un défaut de confidentialité qui peut impacter la vie privée des habitants du logement.

Dans le second exemple (illustration de droite), nous prenons le cas d'un capteur de température qui effectue des mesures de manière périodique et envoie les résultats à l'application de régulation de la température. Ce capteur utilise une valeur particulière pour s'authentifier auprès de l'application. Nous aborderons l'authentification de manière détaillée dans la section 2.5.3. Si cette valeur transite sur le réseau sans protection, un attaquant pourra récupérer cette valeur et l'utiliser pour se faire passer pour le capteur, par exemple. Ici, le défaut de confidentialité entraîne un risque d'usurpation de l'identité d'un équipement qui affecte le niveau de sécurité.

Pour garantir la confidentialité des données, il est courant d'utiliser des méthodes de chiffrement des données et de définir rigoureusement quels utilisateurs ou quels équipements peuvent déchiffrer les données.

2.5.2 *Intégrité*

Pour l'**intégrité**, nous utilisons la définition suivante :

> "[the] absence of improper system alterations." [Avi+04]

L'intégrité peut concerner à la fois les données et les équipements. Dans cette thèse, nous nous sommes focalisés sur l'intégrité des données.

Un risque de sécurité lié à la caractéristique d'intégrité est la modification de données durant une communication. Un exemple de ce type de risque est illustré par la figure 2.7.

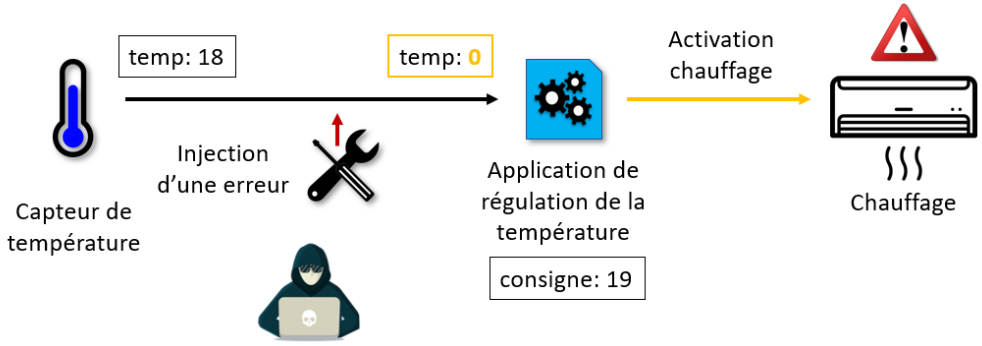

FIGURE 2.7 – Exemple d'altération de données dans le domaine des objets connectés.

Nous sommes, dans cet exemple, en présence d'un système de régulation de la température, dans une habitation ou dans des bureaux. Un capteur de température relève périodiquement la température d'une pièce et transmet cette donnée au système de régulation. Le système possède une température de consigne et ajuste son action sur l'environnement en fonction des données fournies par le capteur. Si la donnée remontée par le capteur est altérée durant son transit, par un acteur malveillant ou par une perturbation environnementale imprévue, le système recevra une donnée erronée et donc une vision erronée de l'environnement. Cela peut amener à des comportements imprévus du système pouvant causer sa mise en panne voire sa dégradation physique [Zet].

Pour garantir et contrôler l'intégrité des données, des protocoles sont mis en place qui utilisent des fonctions de hachage et des mécanismes de signature.

2.5.3 *Authentification et autorisations*

Pour la notion d'**authentification**, nous utilisons la définition suivante :

> « [procédure] qui garantit l'identité d'une personne et de l'origine d'une information. » [Cho09]

Comme indiqué dans la définition, l'authentification peut concerner des personnes physiques ou des systèmes informatiques par le biais des données

qu'ils transmettent. Les systèmes informatiques pervasifs étant principalement composés de systèmes informatiques s'échangeant des données, nous nous sommes concentrés sur ce second point. Dans les systèmes pervasifs, le but est de réaliser de l'authentification entre équipements informatiques de manière aussi autonome que possible, avec un minimum d'interventions humaines.

D'après Dorsemaine *et al.* [Dor+15], il existe trois niveaux d'authentification, présentés ici du plus faible au plus fort en termes de sécurité :

— **nulle** : l'authentification n'est pas utilisée. Il n'existe pas de garantie sur l'identité des équipements.
— **simple** ou **unidirectionnelle** : dans le cadre d'une communication, un des équipements va s'authentifier auprès de l'autre équipement. Il existe une garantie dans un sens de communication mais pas dans l'autre.
— **mutuelle** : les deux équipements vont s'authentifier tour à tour. L'identité des deux équipements est garantie.

La présence d'un système d'authentification permet la mise en place de politiques d'**autorisations** et de **contrôle d'accès**. Ces politiques visent à restreindre les permissions et les droits d'accès à des ressources en fonction des rôles (modèle RBAC[7]) ou des attributs (modèle ABAC[8]) associés à une identité [Far13]. Il existe d'autres modèles étendant ces modèles de base en intégrant des notions de temporalité ou de géolocalisation de l'accès. Quel que soit le modèle choisi, le but est de limiter les impacts en cas de compromission d'un équipement tout en permettant au système de fonctionner.

En l'absence d'authentification, un risque potentiel est l'usurpation d'identité au sein du système. La figure 2.8 présente un exemple de ce type de risque avec l'usurpation de l'identité d'un capteur.

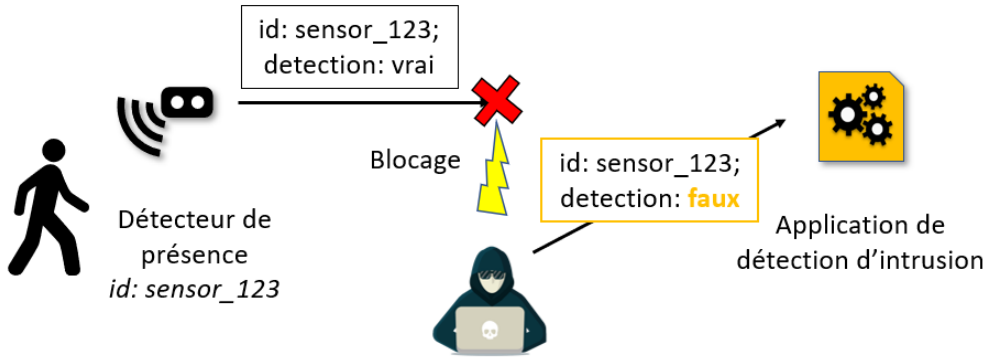

FIGURE 2.8 – Exemple de l'usurpation de l'identité d'un capteur dans un système connecté.

Dans cet exemple, nous considérons une application de détection d'intrusion basée sur des détecteurs de présence. Lorsque le détecteur est déclenché, ce dernier envoie un message à l'application qui peut effectuer les traitements adéquats et déclencher les actions appropriées. Sans protection, un acteur malveillant peut usurper l'identité du détecteur en envoyant des messages à sa place tout en bloquant les messages légitimes. Ce faisant, il devient capable

7. *Role-Based Access Control.*
8. *Attribute-Based Access Control.*

de rendre inutile le détecteur et il crée donc une faiblesse dans le périmètre de détection.

L'authentification se base sur des protocoles utilisant des secrets, avec soit des mots de passe, soit des clés cryptographiques.

Pour résumer, les caractéristiques de sécurité que nous avons présentées, sont rassemblées dans le tableau 2.3. Pour chaque caractéristique, un risque de sécurité est également indiqué.

Caractéristique de sécurité	**Risque**	**Solution**
Confidentialité	Vol de données	Fonctions de chiffrement
Intégrité	Altération de données	Fonctions de hachage et de signature
Authentification	Usurpation d'identité	Utilisation de secrets
Autorisations	Accès non restreint	Contrôle d'accès

TABLE 2.3 – Les caractéristiques de la sécurité étudiés et les risques associés.

Pour aboutir à des solutions pervasives sécurisées, ces caractéristiques doivent être prise en compte dès la phase de conception d'une solution pervasive [HA06].

Nous avons ici présenté les différents risques de sécurité auxquels l'informatique pervasive peut exposer les utilisateurs. Ces risques conduisent à la formulation d'exigences de sécurité à l'égard des solutions pervasives dans leur ensemble : applications pervasives, objets connectés et communications réseau.

2.6 SYNTHÈSE

Dans ce premier chapitre, nous avons présenté le concept d'informatique pervasive qui cherche à rendre des services à des utilisateurs tout en demeurant invisible à leurs yeux ainsi que les trois éléments clés de ce concept : les objets connectés, les applications pervasives et les infrastructures réseau. Nous avons vu que les objets connectés forment un ensemble très hétérogène et que certains objets peuvent être particulièrement contraints, en mémoire de stockage, par exemple. Les objets contraints sont plus difficiles à sécuriser et exigent des solutions de sécurisation légères. Nous avons ensuite vu que le développement des applications pervasives doit répondre à plusieurs exigences : gestion dynamique des ressources, prise en compte du contexte, adaptabilité et sécurité. Pour faciliter le traitement de ces exigences, nous avons étudié l'usage des plateformes pervasives qui permettent de déployer les applications et de leur offrir des fonctionnalités de support comme la gestion de l'hétérogénéité des objets ou la persistance de données. Nous pensons que ces plateformes peuvent également permettre de faciliter la gestion de la sécurité d'une solution pervasive, en gérant, notamment, l'accès sécurisé aux objets. Puis, nous avons présenté plusieurs architectures d'interconnexion entre applications pervasives et objets connectés, basées sur les modèles *cloud* et *fog*.

Enfin, nous avons défini les principales caractéristiques de sécurité que nous allons examiner dans les solutions pervasives : la confidentialité des données, leur intégrité et l'authentification des équipements. La prise en compte de ces caractéristiques permet d'aboutir à des applications sécurisées et de confiance pour les utilisateurs.

Dans le prochain chapitre, nous allons présenter les fonctionnalités de sécurité présentes à plusieurs niveaux d'une solution pervasive : au niveau des plateformes pervasives, au niveau des protocoles de communication et au niveau des objets connectés. L'utilisation adéquate de ces fonctionnalités permet d'obtenir des solutions pervasives possédant les caractéristiques de sécurité que nous avons présenté.

3

PANORAMA DE LA SÉCURITÉ DES ENVIRONNEMENTS PERVASIFS

Nous avons vu dans le chapitre précédent que les solutions informatiques pervasives sont des systèmes complexes composés d'objets connectés, de plateformes sur lesquelles sont déployées des applications pervasives et enfin d'infrastructures réseau pour connecter les éléments précédents. Nous avons également vu que nous pouvions caractériser la sécurité de tels systèmes de manière similaire à un système informatique traditionnel, en examinent comment est géré la confidentialité des données, leur intégrité et l'authentification des équipements au sein d'une solution pervasive. Si ces caractéristiques doivent être considérées de manière globale, les moyens de sécurisation mis en œuvre sont souvent dépendants du sous-système considéré. Les technologies utilisées pour assurer la confidentialité de données transitant sur le réseau et de secrets stockés sur un objet connecté sont par nature différentes.

En partant de la vue globale d'un système informatique pervasif avec une plateforme pervasive que nous avons présentée dans la section 2.3, nous avons identifiés quatre sous-systèmes qui sont présentés dans la figure 3.1.

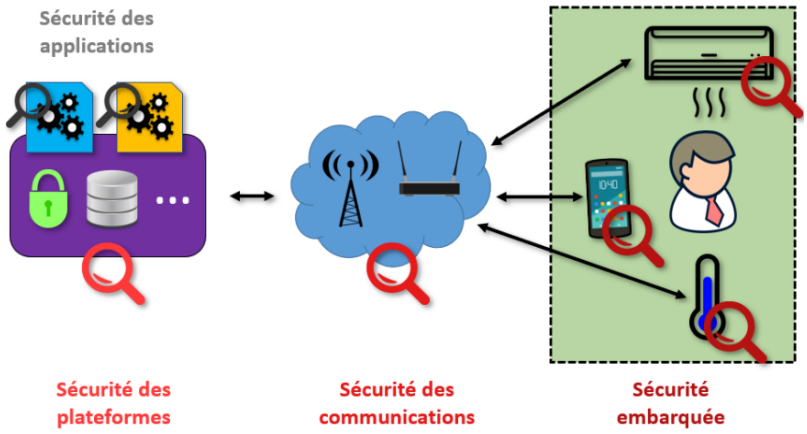

FIGURE 3.1 – Les sous-systèmes à sécuriser dans un système informatique pervasif basé sur une plateforme pervasive.

Ces quatre sous-systèmes sont :

— La **sécurité des applications** concerne le développement sécurisé des applications. Nous n'avons pas traité cette partie dans cette thèse, notre périmètre étant centré autour des objets connectés. Toutefois, il s'agit d'une brique essentielle de la sécurité des solutions pervasives.

— La **sécurité des plateformes** peut désigner deux choses : la sécurité de la plateforme en elle-même, sa résistance aux attaques ; et les fonctionnalités de sécurité qu'elle supporte et dont elle peut faire bénéficier les applications pervasives. Par exemple, le support de différents protocoles sécurisés pour accéder à des objets ou à des services.

— La **sécurité des communications** touche aux échanges réalisés notamment entre les objets connectés et les plateformes pervasives. Ces échanges se déroulent souvent sur des supports non fiables, d'où l'importance de leur sécurisation.
— La **sécurité embarquée** désigne les fonctionnalités de sécurité des objets connectés, destinées à les protéger notamment d'attaques extérieures. Ces attaques peuvent être logiques ou physiques, si les objets sont facilement accessibles.

Dans ce chapitre, nous examinerons les services que proposent plusieurs plateformes pervasives, académiques et commerciales, pour faciliter l'usage d'objets connectés aux interfaces et aux propriétés de sécurité hétérogènes, d'un point de vue fonctionnel et d'un point de vue sécurité. Nous expliciterons ensuite l'hétérogénéité de la sécurité des objets connectés qui complexifie leur utilisation en toute confiance, en examinant plusieurs protocoles de communication et plusieurs technologies de stockage de secrets utilisés dans ce domaine. Nous terminerons ce chapitre par la présentation d'une technologie prometteuse pour la sécurisation à bas coût des objets connectés : les fonctions physiques non clonables.

Nous allons toutefois commencer par présenter quelques notions de base sur les algorithmes cryptographiques de chiffrement et les secrets qu'ils manipulent, car nous allons en rencontrer dans ce chapitre et les suivants.

3.1 ALGORITHMES CRYPTOGRAPHIQUES DE CHIFFREMENT

Il existe deux catégories principales d'algorithmes cryptographiques de chiffrement : les algorithmes basés sur de la cryptographie symétrique et ceux basés sur de la cryptographie asymétrique.

Pour illustrer le fonctionnement général de ces deux catégories d'algorithmes, nous allons utiliser le cas d'usage suivant : protéger la confidentialité d'un message échangé entre deux individus en utilisant du chiffrement. Ainsi, Anne souhaite envoyer un message à Benjamin par le biais d'un canal de communication écouté par Emma, un tiers malicieux. Cette situation est illustrée par la figure 3.2.

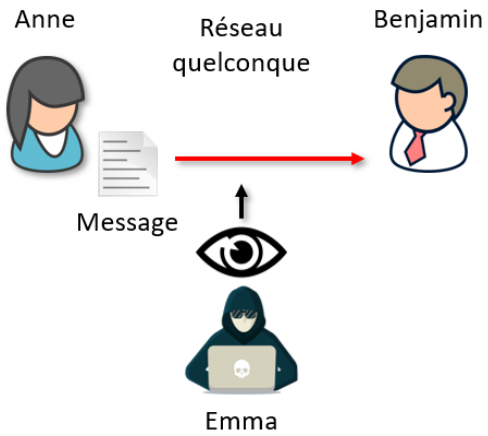

FIGURE 3.2 – Communication d'un message via un réseau quelqueconque.

3.1.1 *Cryptographie symétrique*

En cryptographie symétrique, une clé unique est utilisée pour chiffrer et déchiffrer le message, comme illustré par la figure 3.3. Les algorithmes de cette catégorie possèdent une fonction de chiffrement et une fonction de déchiffrement, qui font usage de cette clé.

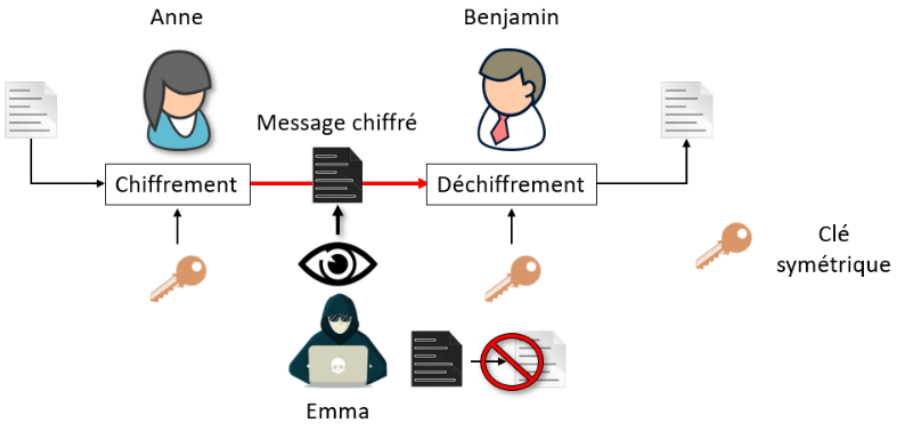

FIGURE 3.3 – Communication d'un message avec du chiffrement symétrique.

Anne est en possession d'un exemplaire de la clé qu'elle utilise avec la fonction de chiffrement de l'algorithme symétrique pour chiffrer les données. Elle transmet ensuite les données chiffrées à Benjamin sur le canal non-sécurisé. Benjamin, pour récupérer les données d'origine, doit également posséder un exemplaire de la clé utilisée par Anne. Il fournit cette clé à la fonction de déchiffrement de l'algorithme symétrique et récupère les données en clair. En écoutant le canal de communication, Emma peut récupérer le message dans sa forme chiffrée. Toutefois, sans la clé symétrique en sa possession, il est impossible pour Emma de retrouver les données en clair.

Les algorithmes symétriques consistent en une série d'opérations linéaires et non-linéaires entre les données d'entrée et la clé, répété un certain nombre de fois. Ainsi une clé symétrique est simplement un vecteur de données d'une taille définie, souvent exprimée en bits. Générer une clé symétrique revient à générer un vecteur de données aléatoires de la taille souhaitée. Si la génération de la clé est vraiment imprévisible, un attaquant n'aura d'autres choix que de tester l'ensemble des clés possibles pour espérer trouver la clé utilisée. Il s'agit d'une attaque par énumération (*bruteforce* en anglais). En choisissant une taille de clé suffisamment grande, supérieure à 128 bits selon les recommandations de l'ANSSI [ANS20], le temps pour tester l'ensemble des combinaisons de clés possibles dépasse les dizaines voire les centaines d'années. Les chances de succès de cette attaque sont alors jugées négligeables.

Les algorithmes symétriques sont réputés pour leurs performances liées aux opérations simples qu'ils utilisent et à la faible taille de leur clé, par rapport aux algorithmes asymétriques que nous présenterons dans la section suivante. Cela en fait un choix idéal pour sécuriser des systèmes disposant de ressources limitées. Un point délicat de ces algorithmes est la distribution des clés. Si Anne, par exemple, génère sa propre clé mais ne la distribue pas, aucun autre

participant ne sera en mesure de déchiffrer ses messages. Il faut alors prévoir des moyens de distribution sécurisés, potentiellement par un canal différent du canal de communication classique. La portée d'action des clés est également une caractéristique importante : elle peut aller d'une clé unique pour l'ensemble des participants d'un réseau de communication à une clé différente par paire de participants. La dernière solution est bien plus robuste en termes de sécurité que la première, car elle permet une isolation des communications des différents participants : en cas de vol d'une clé, seul le lien de communication entre une paire de participants est compromis et pas l'ensemble du système. Toutefois, cette solution est également bien plus complexe à gérer et à faire passer à l'échelle lorsque le nombre de participants augmente, à cause de la croissance exponentielle du nombre de clés.

Des exemples connus d'algorithmes cryptographiques symétriques sont le *Triple Data Encryption Algorithm* (TDEA), standardisé par la RFC 1851 [KMS95], l'*Advanced Encryption Standard* (AES), standardisé par le NIST en 2001 [Dwo+01] et Chacha20, standardisé par les RFC 7539 [NL15], 7634 [Lan+16] et 7905 [Nir15] en 2015. Le niveau de sécurité offert en fonction de la taille de clé utilisée est présenté par le tableau 3.1, basé sur les recommandations émises par l'ANSSI [ANS20] et le NIST [Bar20].

Niveau de sécurité	**Taille de clé (en bits)**	**Algorithme**
Faible	112	TDEA
Modéré	128	AES, Chacha20
Fort	192	AES
Très fort	256	AES, Chacha20

TABLE 3.1 – Niveau de sécurité en fonction de la taille de clé de plusieurs algorithmes de chiffrement symétrique.

3.1.2 *Cryptographie asymétrique*

En cryptographie asymétrique, une clé possède deux parties qui jouent des rôles distincts. Une partie de la clé est dites privée, l'autre partie est dites publique. La partie privée permet d'obtenir la partie publique de la clé, mais la partie publique ne permet normalement pas de retrouver la partie privée d'une clé. Grâce à cette propriété, il est possible de diffuser publiquement la partie publique d'une clé sans la compromettre. Pour faciliter la lecture, nous utiliserons par abus de langage les termes clés privée et clé publique pour désigner respectivement la parties privée et la partie publique d'une clé asymétrique. Chaque algorithme de cette catégorie possède également une fonction de chiffrement et une fonction de déchiffrement, mais ces fonctions n'utilisent pas la même clé comme nous allons le voir dans l'exemple qui suit.

Reprenons notre exemple avec Anne souhaitant envoyer un message à Benjamin, en supposant que Benjamin a généré une clé privée et qu'il a diffusé la clé publique associée. L'échange entre Anne et Benjamin est illustré par la figure 3.4.

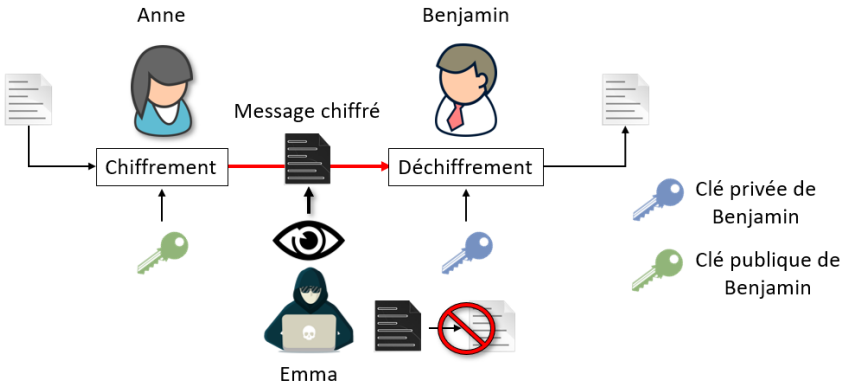

FIGURE 3.4 – Communication d'un message avec du chiffrement asymétrique.

Pour envoyer son message à Benjamin, Anne doit commencer par récupérer la clé publique de Benjamin. Elle utilise ensuite cette clé publique avec la fonction de chiffrement de l'algorithme pour chiffrer le message. Elle peut transmettre le message chiffré à Benjamin. Ce dernier va utiliser sa clé privée avec la fonction de déchiffrement pour récupérer le message d'origine. Emma peut toujours récupérer le message chiffré sur le canal de communication, mais pas le déchiffrer sans la clé privée de Benjamin. Et, contrairement à un algorithme symétrique, le message ne peut être déchiffré avec la clé publique qui l'a chiffré.

Les algorithmes asymétriques se basent sur des problèmes mathématiques algorithmiquement difficiles à résoudre, comme la factorisation de grand nombres premiers ou le problème du logarithme discret. Une clé asymétrique est en fait une collection de paramètres mathématiques définissant l'instance d'un des problèmes mentionnés plus tôt. Ces paramètres doivent être choisis avec soin, car si les problèmes mathématiques évoqués sont difficiles à résoudre dans le cas général, certains cas particuliers peuvent être résolus de manière simple et donc affaiblir la sécurité offerte. Les algorithmes asymétriques sont réputés consommateurs en ressources matérielles, de part la complexité de certaines des opérations mathématiques utilisées et la taille des clés manipulées, qui est supérieure à la taille des clés symétriques à niveau de sécurité équivalent. Pour cette raison, ces algorithmes sont rarement utilisés pour chiffrer l'ensemble d'une communication mais plutôt pour échanger une clé de session entre les participants, c'est-à-dire un secret temporaire utilisable avec un algorithme de chiffrement symétrique, bien plus performant.

La diffusion des clés n'est pas un problème en cryptographie asymétrique. Chaque participant peut générer sa propre clé et diffuser la partie publique au reste des participants. Un autre problème survient cependant : ayant connaissance d'une clé publique donnée, il n'est pas possible de connaître avec certitude le porteur de la clé privée associée. Sachant cela, Emma peut générer sa propre clé et envoyer la partie publique à Anne en se faisant passer pour Benjamin. Il s'agit d'une attaque de type *Man-In-The-Middle* où Emma fait le relai entre Anne et Benjamin tout en pouvant accéder au contenu des messages envoyés. Ce scénario d'attaque est illustré par la figure 3.5.

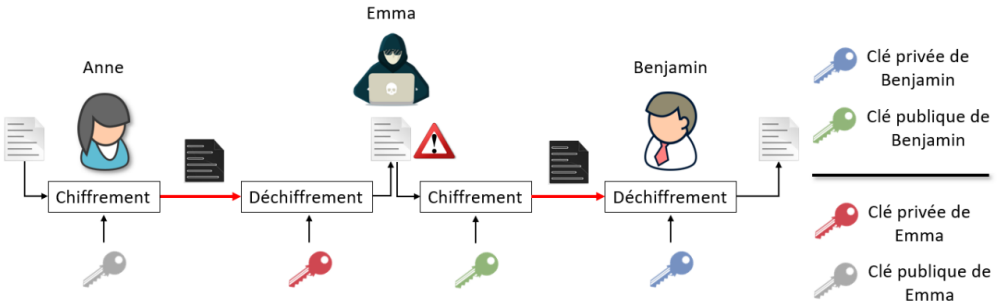

FIGURE 3.5 – Scénario d'attaque *Man-In-The-Middle* lors du chiffrement asymétrique.

Les impacts de sécurité de ce type d'attaque en sont importants : Emma a accès en clair à tous les messages échangés entre Anne et Benjamin, qui n'ont donc plus aucune confidentialité. De plus, Emma peut modifier à loisir le contenu des messages, voire en faire disparaître certains et en créer de nouveaux de toute pièce. Les échanges entre Anne et Benjamin ne sont plus intègres.

Pour éviter ce type d'attaque, un ensemble de standards, d'infrastructures et de processus ont été développés pour apporter de la confiance dans les clés publiques. Il s'agit de l'Infrastructure à Clés Publiques (ICP), que nous présentons dans la section suivante.

Des exemples d'algorithmes asymétriques sont l'algorithme RSA, décrit en 1977 par Rivest, Shamir et Adleman [RSA78], et la cryptographie sur courbes elliptiques, *elliptic-curve cryptography* (ECC) en anglais, suggérée à partir de 1985 à la fois par Koblitz [Kob87] et Miller [Mil86]. Le niveau de sécurité offert par ces algorithmes en fonction de la taille de clé utilisée est présenté par le tableau 3.2, basé sur les recommandations émises par l'ANSSI [ANS20] et le NIST [Bar20].

Niveau de sécurité	**Taille de la clé RSA (en bits)**	**Taille de la clé ECC (en bits)**
Faible	2048	224
Modéré	3072	256
Fort	7680	384
Très fort	15360	512

TABLE 3.2 – Niveau de sécurité en fonction de la taille de clé des algorithmes de chiffrement asymétrique RSA et ECC.

3.1.3 *Infrastructure à Clés Publiques*

Des standards de métadonnées ont été développés pour permettre de lier à une clé publique des informations sur le propriétaire de la clé privée associée. Le standard le plus répandu actuellement est le standard X.509 [Coo+08]. Il définit le format des certificats électroniques qui lient une clé publique aux métadonnées de la clé privée associée. Ces certificats sont validés par des tiers de confiance, appelés autorités de certification (*certificate authority* ou *CA*), via

une signature électronique qui fait également intervenir de la cryptographie asymétrique.

L'Infrastructure à Clés Publiques est un système hiérarchique d'autorités de certification. Une autorité de certification possède un certificat dit racine. Ce certificat sert de base de confiance, il est auto-signé par l'autorité. Un utilisateur doit alors accepter de faire confiance à une autorité de certification en reconnaissant son certificat racine comme valide. L'autorité de certification délivre ensuite des certificats selon les demandes qu'elle approuve. Ces certificats sont signés avec la clé privée de l'autorité, associée à son certificat racine. Chaque certificat indique l'autorité de certification qui l'a validé, ce qui permet à un utilisateur rencontrant un certificat de reconstruire le chemin de certification, c'est-à-dire l'enchaînement des signatures et des certificats jusqu'à un certificat racine. Si le certificat racine appartient à une autorité reconnue par l'utilisateur et que le chemin de certification est correct, alors l'utilisateur peut faire confiance au certificat en bout de chemin.

L'infrastructure à Clés Publiques comprend également des fonctionnalités de gestion du cycle de vie des clés. Chaque certificat possède une période de validité définie par une date de début et une date de fin. Les périodes de validité varient classiquement d'un an à une dizaine d'années, selon le type de certificat. Cela permet de forcer le renouvellement de ces secrets. Un système de révocation existe également, permettant de mettre fin immédiatement à la validité d'un certificat. Un tel système est utile en cas de compromission d'un certificat et d'utilisations malveillantes. Dans les faits, cependant, la révocation repose, notamment, sur la publication de listes de révocation qui doivent être diffusées à large échelle pour rendre la révocation réellement effective.

La cryptographie asymétrique et les Infrastructures à Clés Publiques associées sont aujourd'hui largement déployées et utilisées pour réaliser de l'authentification. Un des exemples de cas d'usage le plus commun est l'authentification réalisée lors de la navigation Web sur des serveurs supportant le protocole HTTPS, comme illustré par la figure 3.6.

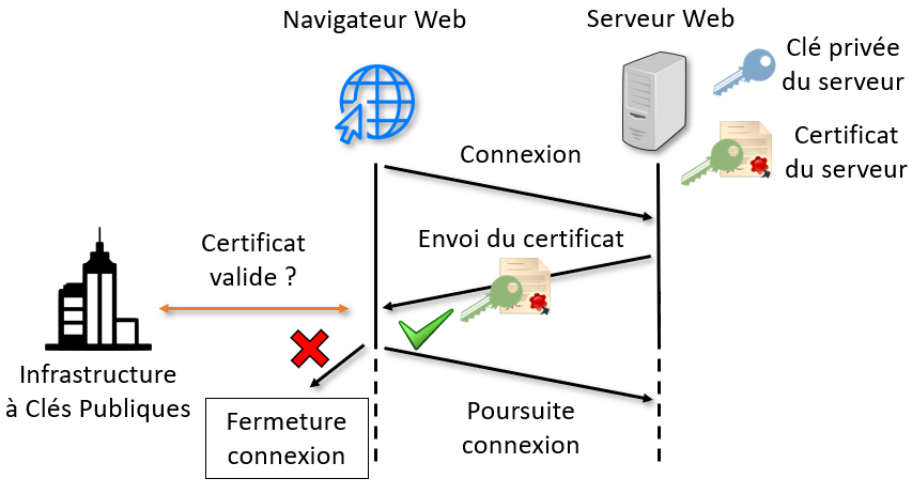

FIGURE 3.6 – Exemple d'authentification d'un serveur Web avec une Infrastructure à Clés Publiques.

Dans cet exemple, la navigateur se connecte au serveur Web et ce dernier lui transmet son certificat. Le navigateur vérifie la validité du certificat en

s'appuyant sur une Infrastructure à Clés Publiques, c'est-à-dire en vérifiant, notamment, que le certificat a bien été délivré par une Autorité de Certification reconnue. Si le certificat est valide, la connexion peut se poursuivre avec l'établissement d'une clé de session entre le navigateur et le serveur pour sécuriser les échanges. Si la vérification du certificat échoue, cela peut être le signe d'une tentative d'attaque *Man-In-The-Middle*. La connexion est alors interrompue.

Cette opération est réalisée de manière transparente par les navigateurs modernes, qui implémentent le protocole sécurisé *Transport Layer Security* (TLS) [Res18], sur lequel s'appuie le protocole HTTPS, et qui possèdent une collection de certificats racines de confiance, communément appelé *truststore*. Cette collection est maintenue à jour régulièrement par les développeurs des navigateurs via des mises à jour logicielles.

Cependant, ce type d'authentification rencontre quelques limites lorsqu'il est appliqué au domaine des objets connectés. C'est particulièrement le cas pour les objets connectés contraints pour plusieurs raisons :

— certaines opérations en cryptographie asymétrique sont coûteuses en ressources matérielles (temps CPU, mémoire vive) en raison de leur complexité. L'opération de déchiffrement de l'algorithme RSA, par exemple, doit effectuer des opérations mathématiques sur de très grands nombres, représentés sur plusieurs milliers de bits. Cela peut affecter les performances ou la durée de vie des objets connectés.
— le déploiement, le maintien et le passage à l'échelle d'une Infrastructure à Clés Publiques impliquent des coûts financiers non négligeables qui peuvent impacter la rentabilité d'une solution connectée, voire la rendre non viable.
— il faut prévoir un processus de renouvellement des certificats. Cela peut concerner le certificat propre à l'objet tout comme les certificats du *truststore* qui permettent à l'objet d'authentifier d'autres équipements.

Après cette introduction sur le fonctionnement haut niveau des algorithmes de cryptographie symétriques, asymétriques ainsi que des Infrastructures à Clés Publiques, nous allons revenir à la sécurité des solutions pervasives en commençant par étudier les fonctionnalités des plateformes pervasives dans la section suivante.

3.2 SÉCURITÉ DES PLATEFORMES PERVASIVES

Nous avons présenté, dans la section 2.3, les plateformes pervasives comme une solution logicielle ayant pour but de faciliter le développement des applications pervasives en offrant des services comme la prise en charge de l'hétérogénéité des objets connectés. Divers acteurs, universitaires et industriels, ont travaillé au développement de plateformes pervasives avec des approches et des fonctionnalités variées allant de la découverte automatisée de services et d'objets [Bar12] à la sélection du meilleur protocole pour communiquer avec un objet [Cap+14]. Nous avions également souligné qu'une plateforme pervasive pouvait jouer un rôle clé dans l'utilisation sécurisée des objets connectés, du fait de son rôle d'intermédiaire entre ces derniers et les applications.

Nous avons alors cherché à savoir quelles fonctionnalités de sécurisation étaient présentes dans les plateformes existantes. Pour ce faire, nous avons

sélectionné un échantillon de plateformes pervasives, en choisissant des plateformes issues de travaux académiques et des plateformes développées par des entreprises pour avoir une vision aussi large que possible. Dans cet échantillon, nous avons étudié la gestion des objets connectés et la prise en compte de la sécurité dans l'accès aux objets.

Pour cette étude, nous avons défini un cadre d'étude proche du schéma général présenté précédemment dans la section 2.3 et illustré par la figure 3.7.

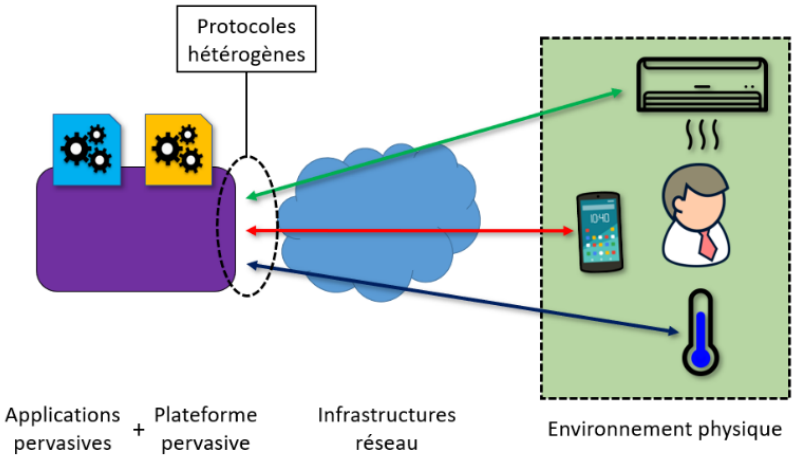

FIGURE 3.7 – Cadre d'étude des recherches sur les plateformes pervasives.

Il comprend un environnement utilisateur contenant un nombre dynamiquement variable d'objets connectés, qui peuvent être ajoutés ou retirés à tout instant. Les objets communiquent avec une plateforme pervasive en utilisant des protocoles de communication qui possèdent des caractéristiques fonctionnelles et des caractéristiques de sécurité hétérogènes, comme nous le verrons dans la section 3.2. Enfin, la plateforme permet le déploiement d'applications pervasives qui doivent utiliser les objets connectés pour fonctionner et rendre des services aux utilisateurs.

Pour comparer les plateformes étudiées, nous avons défini des critères d'étude regroupés en deux catégories : une catégorie fonctionnelle traitant de la gestion des objets connectés et une catégorie sécurité examinant les fonctionnalités déployées en rapport avec les caractéristiques de sécurité présentées dans la section 2.5. Voici le détail de ces critères :

— Catégorie fonctionnelle :
  — **les interfaces de communication prises en charge**. Le domaine de l'IoT se caractérise par une grande hétérogénéité des protocoles de communication. Ainsi, il est souhaitable qu'une plateforme supporte un ensemble varié de protocoles. Il est également appréciable que l'ajout de nouveaux protocoles soit possible pour suivre les évolutions du domaine [ECL14].
  — **l'interfaçage avec les objets**. La manière dont la plateforme présente les objets ou leurs fonctionnalités aux applications pervasives, pour faciliter leur utilisation et masquer leur hétérogénéité.
  — **la gestion du parc d'objets**. La disponibilité des objets peut évoluer dans le temps, en fonction de pannes ou d'opération de maintenance

par exemple. Il est souhaitable que la plateforme puisse gérer ce dynamisme [ECL14].

— Catégorie sécurité :
  — **l'authentification des objets**. Le mécanisme d'authentification permet d'avoir confiance en l'identité des objets et peut servir de base à la gestion de droits d'accès. C'est un point délicat compte tenu de l'hétérogénéité des objets en termes de ressources et de leur potentielle mobilité [Omo+19].
  — **la sécurisation des données en transit**. Selon le type et la criticité des données échangées entre objets et applications, il peut être nécessaire d'en garantir l'intégrité et la confidentialité [Omo+19] dans le but d'assurer le bon fonctionnement d'une application ou de protéger la vie privée des utilisateurs par exemple [ECL14].
  — **la gestion des droits des applications**. Toutes les applications n'ont pas les mêmes besoins d'accès aux objets et aux ressources de la plateforme. Il est souhaitable que des politiques de gestion des droits puissent être déployées pour gérer les permissions des applications et éviter, notamment, des accès conflictuels.

Nous présentons dans la suite de cette section les plateformes que nous avons étudiées, en commençant par les plateformes issues de travaux universitaires, puis les plateformes développées par des entreprises du secteur informatique.

3.2.1 *Plateformes issues de travaux universitaires*

3.2.1.1 *iCasa*

iCasa [1] est une collection d'outils et de services permettant le développement et l'administration d'applications pervasives. C'est un projet du groupe de recherche Adèle [2] du laboratoire d'Informatique de Grenoble, qui travaille dans le domaine de l'ingénierie logicielle, en partenariat avec Orange Labs [3]. Le projet iCasa est porteur de plusieurs contributions présentées, notamment, dans un article de 2014 [Lal+14]. Parmi ces contributions se trouve une plateforme pervasive qui permet le déploiement simple d'applications pervasives et une utilisation aisée par ses applications d'objets connectés et de services de l'environnement. Cette plateforme opère au niveau *fog* et a été testée dans plusieurs scénarios concrets dans les domaines domotique et médical, pour améliorer le soutien à domicile de personnes vulnérables. Elle est divisée en plusieurs modules qui possèdent chacun un rôle propre. Dans notre étude, nous nous sommes concentrés sur les modules dédiés à la gestion des objets connectés. L'agencement de ces modules et leurs liens avec les applications pervasives sont illustrés par la figure 3.8.

1. `http://adeleresearchgroup.github.io/iCasa/snapshot/index.html`.
2. http ://www-adele.imag.fr/.
3. `http://adele.imag.fr/icasa-a-dynamic-pervasive-environment-simulator/index.html`.

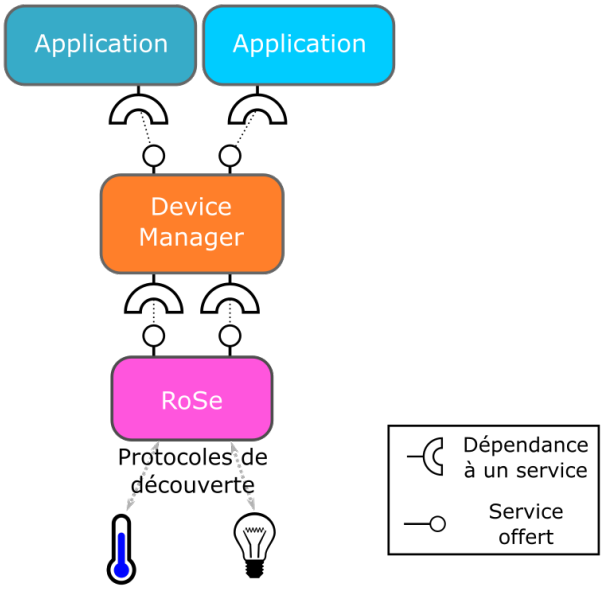

FIGURE 3.8 – Modules de la plateforme iCasa dédiés à la découverte et à la gestion des objets connectés.

La majorité des opérations de gestion et de mise à disposition des objets connectés est géré par le module RoSe, qui est issu de la thèse de Jonathan Bardin [Bar12]. Ce module gère plusieurs protocoles de découverte, qui permettent de chercher dans l'environnement des objets et des services compatibles. Il s'agit d'une fonctionnalité intéressante pour permettre l'intégration d'objets de manière transparente pour les utilisateurs, toutefois il est nécessaire de s'assurer de la sécurisation de cette fonctionnalité pour éviter l'intrusion d'objets ou de services indésirables. Or, nous n'avons pas été en mesure d'identifier les caractéristiques de sécurité concernant l'ajout des nouveaux objets ou de services, comme leur authentification.

Une fois un objet découvert par RoSe, il est réifié sous la forme d'un service avec des propriétés fonctionnelles et, optionnellement, des propriétés non fonctionnelles comme un numéro de version. Les caractéristiques de sécurité supportées par le service comme la confidentialité ou l'intégrité ne semblent pas faire partie des propriétés non fonctionnelles prises en charge, mais le mécanisme proposé semble toutefois suffisamment générique pour gérer les propriétés de sécurité. RoSe maintient à jour un registre des services disponibles, qui est exposé au module de gestion des objets, le *Device Manager*, visible sur la Figure 11. Ce module, développé dans le cadre de la thèse de Rania Ben Hadj [Had18], gère les permissions d'accès aux services. Les services sont ensuite exposés aux applications pervasives. Nous retrouvons dans cette approche les principaux composants d'une architecture orientée services [Pap03] : un registre de services, des fournisseurs de services (les objets) et des consommateurs de services (les applications). Cette architecture permet de profiter, notamment, d'un faible couplage entre les objets connectés et les applications qui ne sont pas en lien direct, mais en interaction indirecte via la consommation de services. La réification des objets sous forme de service permet également de monter en abstraction et facilite la gestion de l'hétérogénéité des objets.

La plateforme propose également des algorithmes de sélection et de substitution de services selon les dépendances fonctionnelles exprimées par les

applications. Cela apporte une certaine capacité d'adaptation pendant l'exécution [ECL14]. Toutefois, ces algorithmes peuvent difficilement se baser sur des critères relatifs à la sécurité pour sélectionner des services, puisqu'ils disposent d'informations partielles à ce sujet : ils ont accès aux propriétés de sécurité déclarées du service, mais pas aux propriétés de sécurité induites par l'objet et le protocole de communication utilisés pour fournir le service.

Lorsqu'une application utilise un service et qu'elle y est autorisée par le module de gestion, le module RoSe se charge de créer une connexion avec l'objet proposant le service via un proxy puis d'exposer le service par le biais des protocoles supportés, tel l'*Hypertext Trasnfert Protocol* (HTTP) et le *Remote Procedure Call* (RPC). Il n'est pas précisé si la connexion établie entre le proxy et l'objet connecté est sécurisée ou non.

3.2.1.2 *PCOM*

PCOM est une plateforme permettant de faciliter la réalisation d'applications distribuées, qui a été présentée, notamment, en 2004 [Bec+04]. C'est un projet de Christian Becker et de son équipe de l'université de Stuttgart. L'architecture de cette plateforme est illustrée par la figure 3.9.

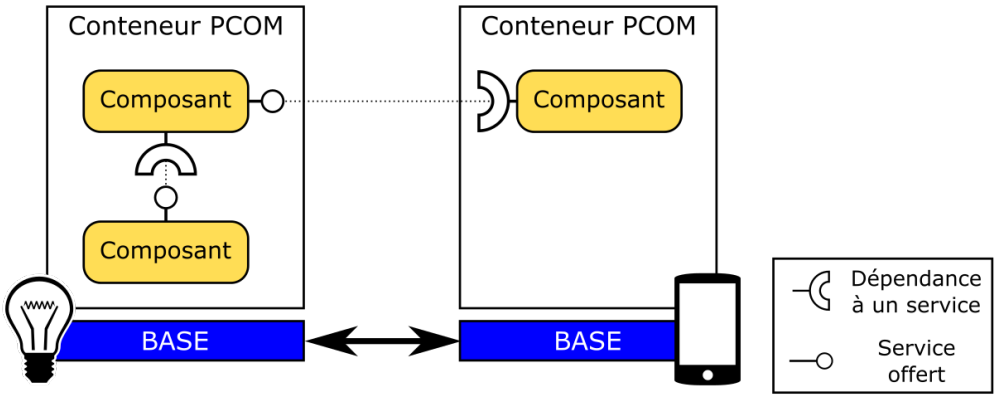

FIGURE 3.9 – Architecture simplifiée de la plateforme PCOM, utilisant le logiciel BASE.

PCOM utilise une approche à composants. Un composant est défini comme une unité logique avec des services et des dépendances explicitement indiquées. Les caractéristiques de sécurité supportées par un service ne semblent pas faire partie de la définition d'un composant. Une application consiste alors en un composant mère et l'arbre de ses dépendances. La plateforme est un conteneur permettant de déployer puis de gérer divers composants. Elle est conçue pour un déploiement au niveau *fog*, sur les objets connectés et les équipements de supervision. Comme illustré sur la Figure 12, un composant donné peut utiliser le service d'un composant présent localement dans le même conteneur que lui, ou bien le service d'un composant distant pour satisfaire une dépendance. Nous retrouvons une approche à service, avec un maillage potentiellement plus fin que la plateforme iCasa mais un objectif de faible couplage et d'abstraction similaire. Chaque conteneur est pourvu de stratégies d'adaptation visant à satisfaire les dépendances de chaque composant, en prenant en compte dynamiquement l'arrivée de nouveaux composants ainsi que l'arrêt de composants existants.

PCOM s'appuie sur le logiciel BASE. Il s'agit d'un précédent projet de C. Becker et de son équipe, qui a pour but de fournir à la couche applicative une interface uniforme d'accès à des services [Bec+03], les fonctionnalités d'un objet, par exemple. BASE dispose d'une architecture modulaire s'articulant autour d'un noyau restreint. Ce noyau gère uniquement la fonctionnalité centrale du logiciel qui est d'accepter et de relayer des requêtes vers des services locaux ou des protocoles de communication. Des fonctionnalités et protocoles peuvent être ajoutés par le biais d'extensions (ou *plug-in*). Nous pouvons supposer que les caractéristiques de sécurité telles que la confidentialité des données transportées par les requêtes, leur intégrité, ainsi que l'authentification des objets et des services distants peuvent également être gérés par l'installation d'extensions appropriées mais cela n'est pas explicitement indiqué.

3.2.1.3 *DiaSuite*

Le projet DiaSuite [4] vise à offrir des outils pour faciliter le développement et le déploiement d'applications pervasives suivant le modèle architectural *Sense/Compute/Control* (SCC). C'est un projet mené par le groupe de recherche Phoenix de l'INRIA dirigé par Charles Consel, et présenté à partir de l'année 2010 [CBC10].

DiaSuite offre des outils pour définir de manière abstraite des applications pervasives et leurs prérequis en matière de capteurs et d'actionneurs. Ces définitions sont utilisées pour générer des structures de base servant à guider l'implémentation des applications. Pour finir, les applications réalisées peuvent être déployées sur une plateforme locale, située au niveau *fog*, qui expose les objets via plusieurs interfaces telles que le *Remonte Method Invocation* (RMI), des *Web Services SOAP* et du *Session Initiation Protocol* (SIP) [Ber+14]. Cela permet une grande versatilité dans l'accès aux objets connectés, cependant la sécurité et le contrôle d'accès de ces interfaces ne sont pas mentionnés.

Le langage de définition DiaSpec permet d'indiquer les interfaces disponibles pour interagir avec chaque objet. Ces interfaces sont ensuite utilisées par les applications pervasives. DiaSuite propose différents mécanismes de communication, quelle que soit la plateforme de déploiement des applications [Ber+14]. Cela permet de gérer l'hétérogénéité des objets connectés, toutefois les caractéristiques de sécurité des différents moyens de communication ne sont pas présentées. Des travaux ont également été menés pour résoudre les problèmes d'accès concurrents aux fonctionnalités d'un objet, un actionneur en particulier. Ces travaux ont permis l'ajout dans DiaSuite d'un système de contexte et de priorité des procédures dans un contexte donné pour éviter l'envoi concurrentiel de commandes mutuellement exclusives à un actionneur [JCL11].

Pour synthétiser, les travaux de recherches que nous venons de présenter étaient innovants lors de leur démarrage et leur réalisation, dans la gestion de l'hétérogénéité des objets connectés ou dans la simplification de l'utilisation des objets via des approches à services. Aujourd'hui, ces projets sont technologiquement daté mais leur architecture orientée services [Pap03] est toujours intéressante pour simplifier l'utilisation des objets connectés par les applications pervasives, grâce à la vision abstraite des objets sous forme de services et le

4. `http://phoenix.inria.fr/software/diasuite`.

faible couplage qu'apporte cette architecture. Nous pensons qu'il est possible d'étendre cette architecture pour y ajouter le support de caractéristiques de sécurité. De plus, la recherche dans ce domaine se poursuit comme l'atteste un article de M. Razzaque *et al.* qui rescensait en 2016 soixante et une propositions de plateformes pervasives [Raz+16] classées en plusieurs catégories, dont l'approche orientée services. Cette enquête nous a fourni des références de plateformes récentes utilisant une architecture orientée service avec des technologies récentes telles que l'architecture REST [Cap+14]. Nous avons sélectionné les plateformes Hydra [ERA10], ubiREST [Cap+14] ainsi que In.IoT [da +21] pour leur proximité avec nos travaux en termes d'architecture.

3.2.1.4 *Autres plateformes*

La plateforme Hydra [ERA10], actuellement connue sous le nom de LinkSmart [5], est conçue pour faciliter le développement d'applications d'intelligence ambiante. Elle utilise une approche orientée services pour assurer l'interopérabilité entre les appareils connectés et les applications. Plusieurs protocoles sont pris en charge pour communiquer avec les objets connectés, tels que le ZigBee ou le Bluetooth. La description des objets s'appuie sur des ontologies connues du milieu IoT telles que OWL et OWL-s [TS19]. Les interfaces de la plateforme sont exposées aux applications pervasives via des technologies Web Services telles que le protocole SOAP [W3C07a]. Concernant la sécurité, l'architecture de la plateforme dispose d'une couche de sécurité qui vise à assurer des caractéristiques de sécurité telles que la confidentialité ou l'authentification. La sécurité repose sur des mécanismes liés aux Web Services, renforcés par des ontologies. Il n'est pas indiqué si ces mécanismes sont compatibles avec les spécifications WS-Security [OAS04], le standard de sécurité de la technologie Web Services. Il n'est pas non plus fait mention de la transmission des informations de sécurité aux applications, pour leur permettre, par exemple, de choisir les services les plus sécurisés.

La plateforme ubiREST [Cap+14] est la continuité des travaux démarrés avec la plateforme ubiSOAP [CRI12]. Le but de ces plateformes est de gérer efficacement plusieurs liens réseau entre des producteurs de services, qui sont généralement des objets connectés, et des consommateurs de services, qui sont généralement des applications pervasives. Pour cela, elles doivent sélectionner le meilleur lien réseau disponible pour échanger des messages entre les producteurs et les consommateurs à un instant donné en utilisant, notamment, une métrique de qualité de service définie par l'utilisateur. A cette fin, ces plateformes sont capables de gérer plusieurs protocoles de communication réseau. La plateforme ubiSOAP utilise le protocole SOAP pour exposer les services disponibles. Il n'est pas fait mention de l'usage de WS-Security ou d'autres mécanismes pour sécuriser les communications. La plateforme ubiREST, quant à elle, utilise un style d'architecture P-REST, pour *Pervasive-REST*, définie par les auteurs comme un raffinement du style d'architecture REST pour les environnements pervasifs. La sécurité des interfaces n'est pas abordée explicitement. Concernant la sécurité des objets connectés, elle est brièvement mentionnée mais de manière trop succincte pour pouvoir connaître précisément les caractéristiques de sécurité supportées.

5. `https://linksmart.eu/`.

Enfin, In.IoT [da +21] est une plateforme visant à faciliter l'enregistrement des objets connectés et permettre leur utilisation de manière performante. Elle utilise une approche orientée services pour réaliser une abstraction des fonctionnalités des objets connectés et faciliter l'interopérabilité. Plusieurs protocoles de communication sont supportés pour communiquer avec les objets connectés, tels que le HTTP ou le MQTT. La sécurité est une préoccupation clé de cette plateforme. Les objets connectés sont authentifiés via des informations qui leur sont fournies par la plateforme lors de leur enregistrement via une liaison réseau. Toutefois, la façon dont sont gérées ces informations d'authentification n'est pas détaillée. En particulier, leur stockage du côté des objets connectés ne semble pas pris en compte. De plus, il n'est pas précisé si les applications pervasives ont accès à des informations de sécurité concernant les services utilisés.

Aujourd'hui, certains des concepts proposés par les plateformes universitaires sont intégrés dans des plateformes développées par des entreprises du secteur numérique, accessibles à grande échelle, et dont les services sont commercialisés.

3.2.2 *Plateformes développées par des entreprises*

3.2.2.1 *Live Objects*

Live Objects[6] est la plateforme pour objets connectés lancée en 2017 par l'entreprise Orange, spécialisée dans les télécommunications. La plateforme permet la connexion d'un grand nombre d'objets, la collecte de données issues de ces objets et leur relayage vers plusieurs services, interne à l'infrastructure d'Orange ou externalisés, comme illustré par la figure 3.10.

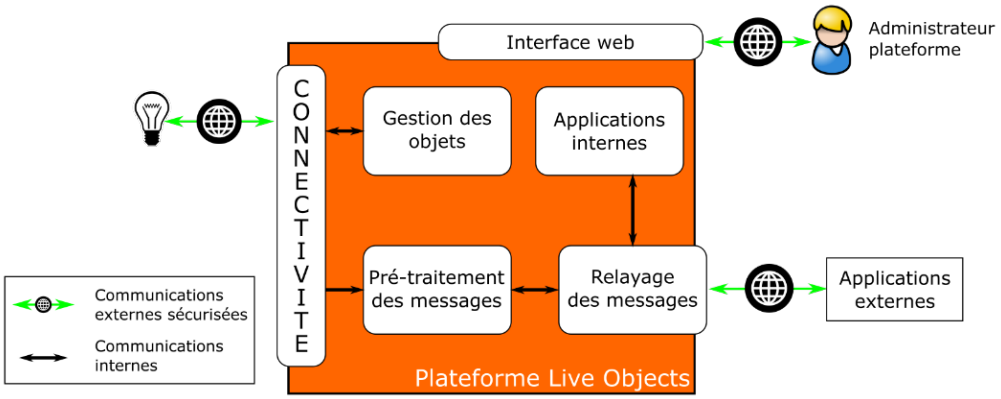

FIGURE 3.10 – Architecture simplifiée de la plateforme Live Objects d'Orange.

La plateforme d'Orange est hébergée sur des infrastructures situées dans le *cloud*. Elle est divisée en plusieurs modules internes :

— Le module **connectivité** permet la connexion entre la plateforme et les objets connectés. Les protocoles l'*Hypertext Trasnfert Protocol* (HTTP) [FR14] et *Message Queuing Telemetry Transport* (MQTT) [And+19] sont supportés pour la communication avec les objets sur le réseau *Internet Protocol*

6. https ://liveobjects.orange-business.com/#/liveobjects.

(IP). Les communications sont sécurisées par la couche réseau *Transport Layer Security* (TLS) [Res18] qui assure la confidentialité de bout en bout, l'intégrité des données et l'authentification de la plateforme. Pour les objets utilisant une interface LoRa, le protocole LoRaWAN est supporté par la plateforme. Ce protocole garantit également la confidentialité et l'intégrité des données en transit, ainsi que l'authentification mutuelle entre l'objet et la plateforme [LoR16].

— La chaîne de modules **pré-traitement des messages** et **relayage des messages** assure la remontée et la distribution des données collectées par les objets. Le premier module permet optionnellement d'appliquer des opérations de décodage ou de mise en forme de la donnée, passage d'un format binaire à un format textuel, par exemple. Le second module permet à l'utilisateur de choisir si les données doivent être envoyées à des applications de traitements internes ou de mises à disposition d'applications externes. Dans ce dernier cas, la plateforme offre des interfaces sécurisées en HTTP ou en MQTT avec l'utilisation de la couche TLS.

— Le module de **gestion des objets** permet d'administrer le parc d'objets. Il est possible d'enregistrer de nouveaux objets, de modifier les propriétés non fonctionnelles déclaratives d'objets existants et de supprimer des objets du parc. Ce module gère également l'envoi de commandes aux objets qui supportent cette fonctionnalité.

L'accès à la plateforme, par un objet connecté ou par une application externe, est authentifié par un secret partagé appelé clé d'API. Chaque clé d'API possède des permissions pouvant restreindre son usage. Par exemple, une clé peut être dédiée uniquement à la remontée de données et ne pourra donc pas être utilisée pour lire des données. Une clé n'est pas nécessairement associée à un objet ou une application de manière unique. Pour les objets connectés, la clé doit être préinstallée dans l'objet avant que ce dernier ne tente de communiquer avec la plateforme. La réalisation sécurisée de cette opération est à la charge de l'utilisateur de la plateforme. Cela suppose la présence d'un stockage sécurisé sur les objets pour protéger la clé. Nous aborderons ce point en détails dans la section 3.3. En l'absence de mesures de sécurité adaptées, un attaquant peut récupérer la clé d'API d'un objet et usurper son identité ainsi que les permissions associées auprès de la plateforme. Cela peut offrir un accès long terme à l'attaquant, car les clés d'API n'ont pas de durée de vie définie lors de leur création. Il est toutefois possible de révoquer une clé d'API pour empêcher ses utilisations futures.

Les données remontées par les objets sont stockées sur la plateforme sous forme de flux de données horodatées. Les données sont visualisables sur l'interface Web et accessibles aux applications possédant une clé d'API avec les permissions requises. Toutefois, aucune information sur les caractéristiques de sécurité des objets qui produisent les données n'est fournie par la plateforme. Si la plateforme collecte des données de plusieurs objets avec des caractéristiques de sécurité hétérogènes, les applications qui utilisent ces données ne peuvent pas le savoir.

3.2.2.2 *Azure IoT*

Azure IoT[7] est la plateforme IoT de l'entreprise Microsoft lancée en 2016, qui s'appuie sur son infrastructure *cloud* Azure. L'offre est découpée en une variété de services allant du système d'exploitation pour objet connecté jusqu'aux applications de traitement des données récupérées. Dans le cadre de nos travaux, nous nous sommes concentrés sur Azure IoT Hub[8], le service permettant la connexion des objets à la plateforme et gérant les communications bidirectionnelles entre les objets et les applications, comme illustré par la figure 3.11.

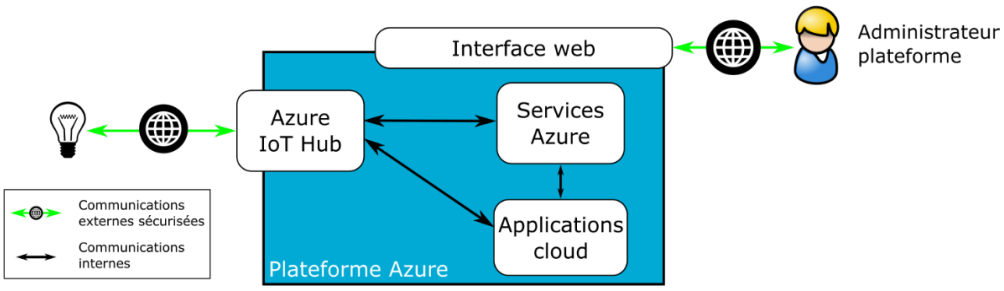

FIGURE 3.11 – Architecture simplifiée de la plateforme Azure de Microsoft.

Le service IoT Hub est le point de contact des objets connectés avec la plateforme. Ce service est en charge d'authentifier les connexions et de les sécuriser pour permettre l'échange de données entre les objets et les applications hébergées sur la plateforme Azure.

Pour authentifier les connexions, le service IoT Hub dispose d'un registre des identités. Il s'agit d'un registre dédié qui stocke l'identité de chaque objet connu de la plateforme ainsi que son moyen d'authentification [Pie+20] : cela peut être un secret partagé ou un certificat. Toute connexion au IoT Hub est mutuellement authentifiée, soit par le protocole TLS dans le cas d'un certificat, soit par un jeton signé dans le cas d'un secret partagé. Une fois la connexion authentifiée, le protocole TLS ou le protocole *Datagram Transport Layer Security* (DTLS) [RTM22] sont utilisés pour garantir la sécurité des échanges.

Concernant la communication entre les objets et la partie applicative, un système de proxy nommé jumeau d'appareil [neh] est utilisé. Pour chaque objet s'étant connecté au moins une fois au hub, un fichier est créé et associé à l'objet. Ce fichier constitue une interface d'échanges entre l'objet concerné et les applications. Le fichier est toujours accessible à la partie applicative, indépendamment de l'état de l'objet qui peut être en veille, par exemple. Lorsque l'objet se connecte à la plateforme, il se synchronise avec le fichier en utilisant un des protocoles gérés par la plateforme, c'est-à-dire HTTP, MQTT ou l'*Advanced Message Queuing Protocol* (AMQP) [OAS].

Ce service de jumeau d'appareil évite à la partie applicative de gérer les problématiques de synchronisation et de gestion de l'état de l'objet ; cependant, il n'offre aucune information sur le délai de prise en compte d'une modification. Pour les applications ayant besoin d'avoir un retour immédiat sur les commandes envoyées à un objet, Azure IoT propose également les méthodes

7. https ://azure.microsoft.com/fr-fr/overview/iot/
8. https ://azure.microsoft.com/fr-fr/services/iot-hub/

directes [phi]. Cette interface permet d'effectuer une requête directe à un objet avec un retour immédiat sur le succès ou non de la requête.

Concernant l'enregistrement de nouveaux objets, Azure IoT propose un service d'enregistrement et de configuration *just-in-time* : le *Device Provisioning Service* [wes]. Le bon fonctionnement de ce service repose sur les prérequis suivants :

— l'objet possède un secret permettant de l'identifier, chargé dans sa mémoire à la fin de sa production, par exemple, et une routine lui permettant de contacter le *Device Provisioning Service* à son démarrage ;
— l'administrateur de la plateforme Azure IoT concernée a récupéré le secret de l'objet et l'a communiqué au service d'approvisionnement.

Si les conditions sont remplies l'objet sera capable lors de son déploiement de s'identifier auprès du service d'approvisionnement. Ce dernier se chargera de l'enregistrer dans un IoT Hub et de lui retourner les informations nécessaires à la connexion au hub. Toutefois, cela demande de prévoir sur l'objet connecté un système de stockage sécurisé du secret initial, puis du secret longue durée obtenue à l'issue de l'étape d'enregistrement. Ce système de stockage doit également supporter la mise à jour des secrets pour, notamment, permettre le renouvellement du certificat de l'objet si cette méthode d'authentification a été choisie. Dans le cas d'un secret partagé, aucune limite temporelle de validité n'est prévue par la plateforme, seulement la révocation.

3.2.2.3 *AWS IoT*

AWS IoT [9] est la plateforme *cloud* proposée par l'entreprise Amazon à partir de 2014 pour les objets connectés. De nombreux services sont proposés par AWS IoT, allant des appareils de terrain au traitement des données sur les serveurs dédiés. Dans le cadre de nos travaux, nous nous sommes focalisés sur le service AWS IoT Core [10], qui gère la connexion des objets à la plateforme AWS, la remontée des données et leur distribution aux services définis, ainsi que l'envoi de commandes comme illustré par la figure 3.12.

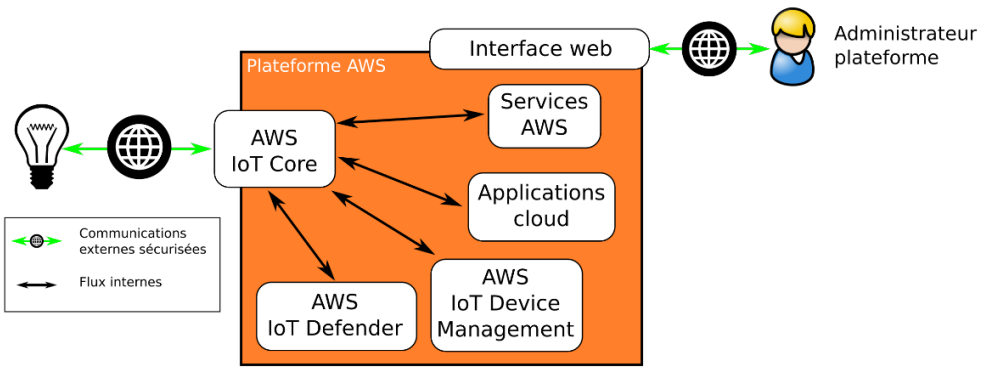

FIGURE 3.12 – Architecture simplifiée de la plateforme AWS IoT.

Ce service supporte des communications sécurisées via les protocoles HTTP et MQTT opérants au-dessus du protocole sécurisé TLS, pour les objets dialoguant

9. https ://aws.amazon.com/fr/iot/
10. https ://aws.amazon.com/fr/iot-core/

sur réseaux IP, et le protocole sécurisé LoRaWAN, pour les objets disposant d'une connectivité LoRa [Pie+20]. Les objets sont authentifiés par certificat, avec un certificat unique délivré à chaque objet par la plateforme d'AWS. Cela implique que les objets doivent disposer d'un dispositif de stockage sécurisé pour conserver la clé privée associée à leur certificat. Les objets doivent également supporter l'installation de nouveaux certificats, dans le cadre du renouvellement de leur certificat, par exemple. Selon la configuration, ces opérations d'installation et de renouvellement sont effectuées automatiquement par la plateforme ou manuellement par l'utilisateur. Côté plateforme, les objets et leurs certificats sont gérés par le module AWS IoT Device Management grâce à un registre dédié [AWSc]. Les interfaces de la plateforme AWS IoT sont également authentifiées par certificat, ce qui permet d'atteindre un niveau d'authentification mutuel.

Concernant la sécurité des objets connectés sur la plateforme AWS IoT, le service AWS IoT Defender permet d'analyser la configuration des objets du parc et certaines de leurs métriques. Par exemple, il est possible de surveiller la quantité de données envoyées à la plateforme. Cela permet de détecter des vulnérabilités ou des comportements malicieux et de lever les alertes correspondantes [AWSd].

Pour enregistrer un nouvel objet auprès de la plateforme, il est possible de le déclarer manuellement dans le service de gestion des objets, puis de générer une clé privée et le certificat unique associé à l'identité de l'objet et, enfin, de charger ces informations dans l'objet. La plateforme AWS supporte également deux procédures d'enregistrement semi-automatisées [AWSb] :

— la première procédure nécessite la collaboration d'un agent sur le terrain, équipé d'une application authentifiée auprès de la plateforme. Lors du déploiement de l'objet, l'agent effectue une demande de certificat temporaire à la plateforme par le biais de l'application. Ce certificat temporaire est transmis à l'objet qui peut l'utiliser pour s'authentifier auprès de la plateforme et s'enregistrer pour obtenir son certificat longue durée. La fuite du certificat temporaire entraîne un risque de détournement modéré du fait de sa courte durée de validité.

— si un agent ne peut être présent pour le déploiement de l'objet, la seconde procédure consiste à charger dans l'objet un secret temporaire. Ce secret doit être partagé avec la plateforme. A sa première connexion, l'objet s'authentifie en utilisant son secret, s'enregistre et récupère un certificat longue durée lié à son identité. La fuite du secret temporaire entraîne un risque de détournement fort ; car un secret valide peut théoriquement enregistrer autant d'objets que souhaité. De plus, la désactivation d'un secret n'affecte pas les objets enregistrés avant la désactivation.

Concernant les interactions entre les objets et les applications, AWS IoT Core propose l'interface *Device Shadow* [AWSa]. Le principe est similaire au service de jumeau d'appareil de la plateforme Azure. Pour chaque objet se connectant à la plateforme, le service IoT Core sauvegarde dans un fichier propre à l'objet les données remontées sous forme de couples clé/valeur. Les applications ont accès en lecture et en écriture à ce fichier. Lorsque le fichier est modifié par une application, IoT Core se charge de transmettre les modifications à l'objet à sa prochaine connexion à la plateforme. Cela permet aux applications de ne pas

avoir à gérer les problématiques de connectivité avec l'objet, ce qui facilite le développement de ces applications.

3.2.3 *Synthèse*

Les résultats de notre étude sur les plateformes pervasives sont présentés de manière synthétique dans le tableau 3.3, suivant les critères indiqués en début de section.

		iCasa/RoSe	**PCOM/BASE**	**DiaSuite**
FONCTIONNEL	**Interface(s) de communication**	HTTP, RPC	Non indiqué	RMI, SOAP, SIP
	Interfaçage avec les objets	Interface normalisée, proxy	Interface normalisée	Interface normalisée
	Gestion du parc d'objets	Registre de services	Registre de services, registre des appareils	Protocole de découverte
SECURITE	**Authentification des objets**	Non indiqué	Non indiqué	Non indiqué
	Sécurisation des données en transit	Non indiqué	Non indiqué	Non indiqué
	Gestion des droits	Non indiqué	Non indiqué	Non indiqué

		Live Objects	**Azure IoT**	**AWS IoT**
FONCTIONNEL	**Interface(s) de communication**	HTTP, MQTT, LoRaWAN	HTTP, MQTT, AMQP	HTTP, MQTT, LoRaWAN
	Interfaçage avec les objets	Flux de données, commandes directes	Clone d'appareil, commandes directes	Clone d'appareil
	Gestion du parc d'objets	Registre des appareils	Registre des identités	Registre des appareils
SECURITE	**Authentification des objets**	Certificat, secret partagé	Certificat, secret partagé	Certificat
	Sécurisation des données en transit	TLS, LoRaWAN	TLS	TLS, LoRaWAN
	Gestion des droits	Mot de passe, Permissions	Mot de passe, Permissions	Mot de passe, Permissions

TABLE 3.3 – Synthèse des recherches sur les plateformes pervasives.

Cette synthèse nous permet d'identifier des différences d'approches entre les plateformes universitaires et les plateformes proposées par les entreprises, notamment sur les points suivants :

— **interfaçage avec les objets**. Les plateformes universitaires proposent toutes des interfaces uniformes pour faciliter l'utilisation des objets et masquer leur hétérogénéité. En revanche, les plateformes proposées par des entreprises sont plus axées sur la fourniture de données dont la structuration peut varier, via les fonctionnalités de clone d'appareil. Cette dernière approche masque moins l'hétérogénéité des objets connectés et complexifie l'utilisation d'objets diversifiés dans une solution pervasive.
— **gestion sécurisée des objets connectés**. Les travaux universitaires étudiés abordent peu le sujet de la sécurité dans les opérations de gestion et la communication avec les objets connectés. Ces mêmes sujets sont aujourd'hui mis en avant par les entreprises qui développent et commercialisent leurs plateformes, pour gagner la confiance des utilisateurs. Nous constatons sur ces plateformes l'utilisation de méthodes standards qui sont réputées sécurisées, mais qui peuvent ne pas convenir complètement à certains objets utilisés dans les solutions pervasives, à cause des contraintes mémoires et énergétiques de ces derniers (voir la section 2.2 à ce sujet). De plus, ces méthodes de sécurisation supposent que les objets sont capables de stocker de manière sécurisée un secret ou un certificat, ce qui n'est pas toujours garanti comme nous le verrons dans la section 3.4.

Pour finir, dans toutes les plateformes que nous avons étudiées, nous remarquons que les applications n'ont pas accès à des informations sur la sécurité associée aux données ou aux services mis à disposition. Elles ne peuvent pas différencier des données ou services issus d'objets très sécurisés des données ou services fournis par des objets peu sécurisés, ce qui peut entraîner des risques de sécurité ou de confiance dans les données.

Les protocoles de communications, par exemple, peuvent posséder des configurations de sécurité très variées comme nous allons le voir dans la section suivante.

3.3 SÉCURITÉ DES COMMUNICATIONS

Après avoir étudié la sécurité des plateformes, nous nous sommes intéressés aux protocoles de communication utilisés entre les plateformes et les objets connectés. Nous nous sommes concentrés sur les protocoles sans fil, qui sont particulièrement populaires dans le domaine des objets connectés grâce aux faibles contraintes qu'ils présentent par rapport aux protocoles filaires, en termes d'infrastructures, par exemple. Mais ces protocoles présentent également une surface d'attaque bien plus grande. C'est pourquoi nous avons étudié les fonctionnalités permettant l'authentification des équipements, la confidentialité et l'intégrité des données au sein de quatre de ces protocoles, choisis parmi les plus utilisés : le protocole ZigBee, le protocole Bluetooth Low Energy, le protocole LoRaWAN et, pour finir, le protocole Wi-Fi.

Nous avons étudié plusieurs protocoles de communication principalement destinés aux objets connectés avec comme but d'évaluer la qualité des fonctionnalités de sécurité de ces protocoles, et de déterminer leurs points communs et leurs différences dans la sécurisation des communications. Il a fallu réfléchir à un système d'équivalence pour permettre la comparaison au sein de systèmes multi-protocolaires, comme les environnements pervasifs. Nous avons aussi pris

en compte la complexité des configurations de sécurité possibles et le niveau d'expertise requis pour aboutir à une configuration sécurisée. Enfin, nous avons cherché pour chaque protocole s'il possédait des vulnérabilités connues pour, notamment, montrer que la sécurité n'est pas une notion figée et que le niveau de sécurité d'un protocole donné peut grandement varier lors de la découverte de nouvelles vulnérabilités.

3.3.1 *ZigBee*

Le protocole Zigbee est une solution de connectivité IoT gérée par la Connectivity Standards Alliance [11] (anciennement la Zigbee Alliance), qui développe également d'autres technologies et standards dans le domaine de l'IoT.

ZigBee est présenté comme une solution IoT complète permettant la communication d'objets connectés, de l'organisation du réseau jusqu'à la structuration des données échangées entre les objets. Le protocole est adapté à des cas d'usage domotiques et industriels tels que la gestion d'éclairages ou la collecte des données de capteurs de terrain.

La conception du protocole remonte à l'année 1998 et la version 1.0 du standard est publiée en 2005. Le protocole a ensuite connu plusieurs évolutions qui ont, notamment, concernées les fonctionnalités de sécurité du protocole. Cela signifie que plusieurs versions du protocole peuvent exister sur le terrain, avec des fonctionnalités de sécurité différentes. Il est donc important de connaître la version du protocole utilisée par un équipement en vue de son intégration à une solution pervasive et de privilégier, autant que possible, la version la plus récente lors de la conception d'une nouvelle solution : la version ZigBee 3.0, publiée en 2015, au moment de la rédaction.

Le protocole ZigBee est structuré en réseaux, c'est-à-dire des ensembles d'équipements qui communiquent entre eux. Nous appelons nœud un équipement qui supporte le protocole ZigBee et peut donc rejoindre un réseau ZigBee. Chaque nœud assume un rôle dans le réseau parmi trois possibles. Les rôles font varier les fonctionnalités accessibles à un nœud dans le réseau, mais aussi les ressources consommées pour opérer le protocole. Ainsi, il est possible d'attribuer aux objets les plus contraints le rôle le plus basique, qui suffit pour accomplir leur tâche. Les trois rôles possibles, du plus limité au plus complet, sont :

— *ZigBee End Device* (ZED) est le rôle le plus basique, celui d'un simple participant à un réseau ZigBee avec des fonctionnalités permettant de rejoindre un réseau déjà existant et de communiquer sur un réseau une fois intégré. C'est le rôle adapté pour un capteur par exemple.
— *ZigBee Router* (ZR) possède toutes les fonctionnalités d'un *End Device* et permet en plus de relayer des messages à d'autres équipements ZigBee, augmentant potentiellement la portée d'un réseau ZigBee. Ce rôle nécessite plus de ressources qu'un *End Device*.
— Enfin, *ZigBee Coordinator* (ZC) est le rôle le plus fourni : il s'agit du rôle qui gère l'existence d'un réseau ZigBee et ses paramètres. Il a notamment une fonctionnalité de *Trust Center* permettant la gestion de la politique d'accès au réseau ainsi que d'autres politiques de sécurité. Chaque réseau

11. https://csa-iot.org/

ZigBee possède au plus un nœud avec le rôle de *Coordinator*. Ce rôle est généralement associé à un équipement dédié, une passerelle par exemple.

Différentes topologies réseaux sont supportées par le protocole pour s'adapter à divers cas d'usage : le réseau en étoile pour un réseau centralisé autour d'un nœud *Coordinator* ou le réseau sous forme d'arbre qui étend la topologie précédente en ajoutant des nœuds *Router*, par exemple. Ces topologies sont illustrées par la figure 3.13.

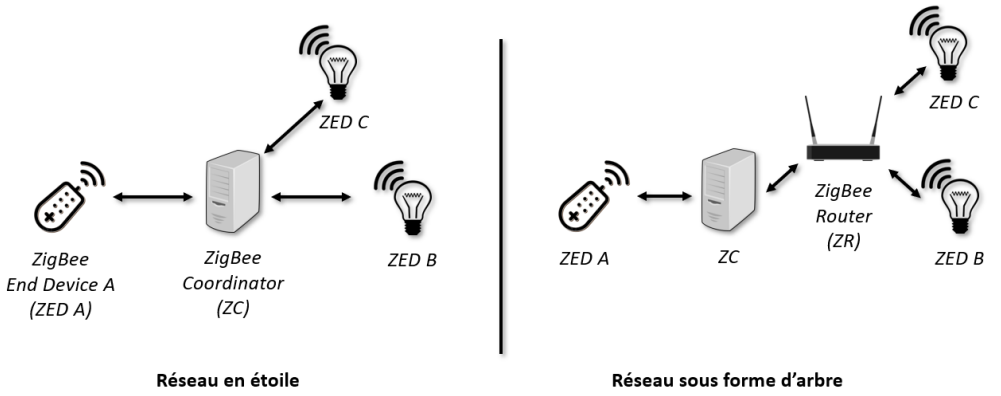

FIGURE 3.13 – Différentes topologies réseaux supportées par le protocole ZigBee.

Pour rejoindre un réseau ZigBee, un nouveau nœud doit obtenir la *Network Key* associée à ce réseau. Il s'agit d'un secret connu par tous les nœuds participants au réseau et qui est utilisé pour sécuriser les messages échangés, en assurant leur confidentialité et leur intégrité. Pour récupérer ce secret, plusieurs méthodes existent :

- le **chargement manuel**. Si le secret est connu et que le nouvel objet peut accepter ce paramètre en entrée, il est possible de le charger manuellement par le biais d'une action de l'utilisateur. Dans ce cas, il revient à l'utilisateur d'authentifier l'objet dans lequel il charge le secret. Il est aussi nécessaire de restreindre la diffusion du secret pour éviter l'ajout non contrôlé d'objets. Cette méthode est toutefois peu compatible avec des solutions pervasives car elle demande une intervention directe de l'utilisateur pour manipuler un secret.
- la **récupération automatique**. Un nouvel objet peut également découvrir par lui-même un réseau ZigBee et effectuer une requête pour le rejoindre. Plusieurs cas sont alors possibles :
  - Si le réseau est ouvert, n'importe quel objet effectuant une demande se verra communiquer la *Network Key* pour rejoindre le réseau. Aucune authentification n'est alors effectuée. De plus, la *Network Key* est communiquée en clair au nouvel objet qui ne fait pas encore partie du réseau. Cette méthode facilite grandement l'intégration de nouveaux objets au réseau, mais n'offre aucune authentification des objets.
  - A partir de ZigBee 3.0, l'objet peut disposer d'un code d'installation, c'est-à-dire un secret unique chargé dans sa mémoire à sa fabrication. Ce code doit être transmis manuellement au nœud *Coordinator* gérant le réseau avant que l'objet ne tente de rejoindre le réseau. Il est utilisé pour dériver une clé unique entre le nouvel objet et le nœud *Coordinator*, qui est ensuite utilisée pour transmettre la *Network Key* de

manière confidentielle et intègre. Cette méthode permet d'effectuer une authentification mutuelle entre le nœud *Coordinator* et le nouveau nœud, basée sur la connaissance du code d'installation. Toutefois, l'intervention d'un utilisateur reste nécessaire.

— Si l'objet ne dispose pas de code d'installation, il peut contacter le nœud en charge du réseau avec un secret défini par le standard dont la valeur est « ZigBeeAlliance09 ». Ce secret étant publiquement connu, le niveau de sécurité offert par cette méthode est faible.

L'ajout d'un nouveau nœud est donc une opération qui peut se révéler non triviale et qui peut aboutir à plusieurs niveaux d'authentification et de confiance. Une fois sur un réseau, un nœud va manipuler deux types de secrets pour sécuriser ses échanges :

— La *Network Key*, qui lui donne l'accès au réseau. Cette clé permet de sécuriser les messages nœud à nœud. Un message envoyé sur un réseau ZigBee est chiffré avec cette clé, mais il est également déchiffré par tout nœud du réseau qui doit relayer le message.

— La *Link Key*, qui permet de protéger des messages vers un destinataire précis. Pour cela, il faut générer et partager une *Link Key* par paire de nœuds qui doivent communiquer de manière sécurisée de bout en bout. Ce mécanisme est notamment utilisé entre un nouveau nœud et celui qui gère le réseau pour sécuriser la transmission de la *Network Key*.

Ces deux types de secrets sont des secrets de longue durée qui doivent être stockés dans la mémoire des équipements. La sécurité du stockage de ces secrets conditionne la sécurité offerte par le protocole. Par exemple, si la *Network Key* est récupérée par un attaquant, c'est la confidentialité de l'ensemble des communications du réseau qui est potentiellement compromise.

Ces secrets sont utilisés avec des algorithmes cryptographiques pour assurer la confidentialité et l'intégrité des échanges réseaux [Rud17]. Pour le chiffrement des données, le protocole ZigBee utilise l'algorithme AES en mode CTR avec une taille de clé de 128 bits. C'est un choix cohérent avec un objectif de faible consommation de ressources ; car c'est un algorithme symétrique, qui peut être implémenté de manière efficiente. De plus, une taille de clé de 128 bits offre un niveau de sécurité adéquat, en phase avec les recommandations d'organismes tels que l'ANSSI [ANS20] ou le NIST [Bar20].

Pour l'intégrité et l'authentification des messages, l'algorithme AES en mode CBC-MAC est utilisé, avec également une taille de clés de 128 bits.

Des chercheurs en sécurité, comme J. Wright [Wri09] ou T. Zillner *et al.* [ZS15], ont révélé des vulnérabilités et des configurations du protocole ZigBee pouvant réduire significativement la sécurité offerte. Les principales vulnérabilités concernent la gestion des clés sur laquelle repose la sécurité du protocole, comme cela est précisé dans la spécification même de ZigBee [Zig15] :

> The level of security provided by the ZigBee security architecture depends on the safekeeping of the symmetric keys [...]. Trust in the security architecture ultimately reduces to trust in the secure initialization and installation of keying material and to trust in the secure processing and storage of keying material.

Or, l'installation des clés sur un nouvel appareil est une opération délicate. Il existe principalement deux méthodes :

— la pré-installation des clés, notamment la *Network Key*, avant le déploiement de l'objet. Cela signifie souvent que les clés sont statiques et que tout changement de clé conduit à devoir réécrire une partie voire toute la mémoire morte de l'appareil. De plus, si l'appareil est exposé physiquement, une attaque matérielle peut permettre l'extraction des clés stockées dans la mémoire [Wri09].
— l'installation *over-the-air*, c'est-à-dire via le protocole ZigBee lui-même lorsque l'appareil demande à rejoindre le réseau. Ce mécanisme offre plus de flexibilité mais il est souvent vulnérable : pour l'installation de la *Network Key* par exemple, il a été montré que dans certaines configurations elle est transmise en clair sur le réseau sans fil [Wri09] tandis que, dans d'autres configurations, elle est transmise en utilisant une clé fixe et connue [ZS15]. Dans ces deux cas, elle est alors facilement récupérable par un attaquant écoutant passivement le réseau.

De plus, la fin de vie des appareils et l'impact sur les secrets ne semble pas prise en compte : il n'existe pas dans ZigBee de mécanisme de révocation de clé. Il existe un mécanisme de rotation de clé, mais il semble peu utilisé en pratique [ZS15].

Pour ces raisons, la mise en place et l'évolution d'un réseau ZigBee de manière sécurisée est une tâche complexe. Évaluer le niveau de sécurité effectivement offert par un réseau ZigBee demande de prendre en compte plusieurs paramètres : la version du protocole utilisée, les méthodes d'ajouts de nouveaux objets autorisées et la manière dont sont gérées et conservées les clés utilisées.

3.3.2 *Bluetooth Low Energy (BLE)*

Le *Bluetooth Low Energy*, communément abrégé BLE, est un des protocoles spécifiés par le Bluetooth Special Interest Group [12], une communauté mondiale de plus de 36 000 entreprises, qui fait évoluer la spécification Bluetooth et gère la certification des produits utilisant des technologies Bluetooth pour garantir leur interopérabilité et leur conformité.

Le BLE a été conçu pour des scénarios d'objets connectés contraints en énergie, qui transmettent de faibles volumes de données de manière intermittente [Wu+20]. Un scénario d'usage classique du protocole est la communication entre un téléphone portable et un capteur connecté.

La première version du protocole est sortie en 2009, dans la spécification Bluetooth 4.0, puis le protocole a connu plusieurs évolutions. Certaines de ces évolutions ont directement concernées les fonctionnalités de sécurité du protocole. Comme pour le protocole ZigBee, cela signifie que la version du protocole BLE utilisé par une solution donnée peut avoir un impact sur la sécurité du système. Lorsque cela est possible, il faut privilégier les versions les plus récentes qui disposent d'un plus large choix d'options de sécurité.

Pour échanger des données entre équipements plusieurs modes de connexion sont définis par la spécification, avec des caractéristiques de sécurité différentes : le mode *broadcast* et le mode *connections*.

12. https ://www.bluetooth.com/

Le mode *broadcast* est très simple et ne propose pas de fonctionnalité d'authentification ou de confidentialité. Dans ce mode, un appareil émet de manière périodique des paquets. Il agit en tant que *Broadcaster*. Tout autre appareil à portée peut recevoir et lire ces paquets, il est nommé *Observer*. C'est un mode de communication unidirectionnel, car aucun mécanisme de réponse n'est défini. Toutefois, un appareil peut assumer les deux rôles présentés précédemment pour recevoir des paquets et en émettre soi-même. Ce mode est souvent utilisé par les appareils pour signaler leur présence, puis basculer sur le second mode de communication.

Le mode *connections* permet une connexion bidirectionnelle point à point entre deux appareils, avec un paradigme client/serveur. Un appareil client va initier une connexion vers un appareil serveur afin d'accéder à une ou plusieurs données. A l'ouverture d'une connexion, les appareils doivent définir/négocier plusieurs paramètres, notamment le niveau de sécurité de la connexion. Ce dernier peut conditionner l'accès à certaines données, qui ne peuvent être accessibles qu'avec un niveau suffisamment élevé par exemple. La spécification du BLE définie deux *security modes* et pour chaque mode, plusieurs niveaux de sécurité. Les modes de sécurité sont :

— Le *Security Mode 1,* qui gère le chiffrement du lien de communication pour garantir la confidentialité des données échangées. Un total de quatre niveaux sont disponibles pour ce mode. Ils sont présentés dans le tableau 3.4.

Niveau de sécurité	**Description**
Level 1	Aucune sécurité
Level 2	Chiffrement non authentifié
Level 3	Chiffrement authentifié
Level 4	Chiffrement authentifié avec ECDH [13]

TABLE 3.4 – Description des niveaux de sécurité du Security Mode 1.

Pour le chiffrement des données, l'algorithme AES en mode CCM est utilisé avec des clés de 128 bits.

— Le *Security Mode 2,* qui gère la signature de données pour garantir leur intégrité et leur authenticité. Deux niveaux sont disponibles pour ce mode. Ils sont présentés dans le tableau 3.5.

Niveau de sécurité	**Description**
Level 1	Signature sans authentification
Level 2	Signature avec authentification

TABLE 3.5 – Description des niveaux de sécurité du Security Mode 2.

Il est possible de combiner les deux modes présentés précédemment pour bénéficier du chiffrement et de la signature des données grâce au *Mixed Security Mode*. L'ensemble de ces options offre une vaste palette de niveaux de sécurité

hétérogènes, allant d'échanges sans confidentialité ni intégrité à des échanges confidentiels et intègres, mais induit également une certaine complexité dans le choix du niveau adapté pour un type de donnée.

Une étape clé dans la sécurisation d'une communication en mode *connections* est l'appairage (*pairing* en anglais). Cette phase permet d'initier la communication entre deux équipements. Durant l'appairage, les équipements peuvent s'authentifier et mettre en place des clés de sessions et des secrets long terme. Plusieurs méthodes d'authentification sont prévues par la spécification BLE. Le choix de la méthode utilisée repose en partie sur les capacités d'entrée/sortie des équipements impliqués (présence d'un écran, d'un clavier...) et sur le niveau de sécurité souhaité pour la communication. La version du protocole utilisée a également son importance.

Prenons l'exemple de la méthode d'appairage la plus basique proposée par la spécification, la méthode Just Works™. Cette méthode ne nécessite pas d'intervention de l'utilisateur, ce qui en fait une méthode de choix pour une solution pervasive. Toutefois, cette méthode possède deux variantes aux caractéristiques de sécurité différentes, selon la version du protocole BLE utilisée :

— la variante *Legacy Connections*, de la version 4.0 à la version 4.2 de la spécification Bluetooth. Dans cette variante, une valeur fixe et publiquement connue est utilisée pour dériver la clé de session entre les appareils. Cette méthode n'est donc pas sécurisée.

— la variante *Secure Connections*, à partir de la version 4.2 de la spécification. Ici, l'algorithme ECDH sans authentification est utilisé pour convenir d'une valeur commune, qui est ensuite utilisée pour dériver la clé de session. Cette méthode protège de l'écoute passive du média de communication ; car la valeur n'est jamais échangée sur ce dernier et n'est pas connue à l'avance. Cependant, l'absence d'authentification rend cette méthode vulnérable aux attaques de type *Man-In-The-Middle*, ce qui la rend faiblement sécurisée.

D'autres méthodes d'apparaige plus poussées sont décrites dans la spécification Bluetooth et permettent d'atteindre un niveau d'authentification mutuelle entre les appareils. Cependant, ces méthodes nécessitent la présence d'équipements particuliers sur les appareils, un clavier par exemple, et l'intervention d'un utilisateur. Pour ces raisons, elles sont moins adaptées aux solutions pervasives.

D'un point de vue technique, le chiffrement des données est assuré par l'algorithme AES en mode CTR avec une taille de clé de 128 bits. L'intégrité et l'authentification des données reposent sur l'algorithme AES en mode CBC-MAC avec une taille de clé de 128 bits également. Ce sont des choix identiques au protocole ZigBee et qui sont en cohérence avec les objectifs de niveau de sécurité et d'efficacité du protocole.

Pour la signature des données, l'algorithme AES en mode CMAC [Dwo+01] est utilisé avec des clés de 128 bits. C'est là aussi un choix cohérent avec les objectifs de sécurité du protocole.

Plusieurs recherches ont été menées sur la sécurité du protocole BLE [Wu+20 ; Cay+21 ; San+19] et ont soulignées des faiblesses dans certaines parties du protocole. Par exemple, sur la négociation de clés [ATR20] ou sur le mécanisme de reconnexion [Wu+20]. La présence de ces vulnérabilités peut grandement

affaiblir le niveau de sécurité d'un objet. Cela illustre le côté dynamique de la sécurité et l'importance de rester informé sur les dernières vulnérabilités découvertes.

Pour synthétiser le protocole Bluetooth Low Energy (BLE) définit divers canaux de communications avec un large éventail de niveaux de sécurité. Ces niveaux de sécurité permettent d'assurer de manière plus ou moins complète l'intégrité et la confidentialité des données échangées, ainsi que l'authentification des appareils engagés dans une communication. Le niveau de sécurité offert est ainsi très hétérogène et peut être très différent d'un appareil à un autre, en fonction des choix d'implémentation et des configurations choisies.

3.3.3 *LoRaWAN*

Le LoRaWAN est une technologie de communication standardisée par la LoRa Alliance [14], une organisation à but non lucratif, créée en 2015, pour supporter l'évolution du standard. Elle propose également un programme de certification des équipements pour assurer leur interopérabilité et tester leur conformité.

Le LoRaWAN est un protocole de communication sans fil destiné aux réseaux longue portée. Le protocole est pensé pour permettre la communication avec des cibles contraintes, il est donc optimisé pour être peu couteux à déployer et peu énergivore. Les cas d'usage principaux sont la communication IoT dans des villes connectées ou des sites industriels de grande taille.

Le protocole connaît sa première version en janvier 2015. C'est un protocole récent qui a été conçu en pleine conscience des enjeux de sécurité et d'efficacité énergétique propres au domaine des objets connectés notamment. Plusieurs mises à jour du protocole ont ensuite vu le jour. Ces mises à jour ne concernent pas, pour le moment, les fonctionnalités sécurité du protocole, qui restent identiques à travers les différentes versions du protocole.

Un réseau LoRaWAN est géré par un équipement dédié appelé *Network Server*. Cet équipement est, notamment, en charge de la vérification des données en provenance des nœuds de terrain, c'est-à-dire des objets connectés. L'architecture générale d'un réseau est présentée par la figure 3.14.

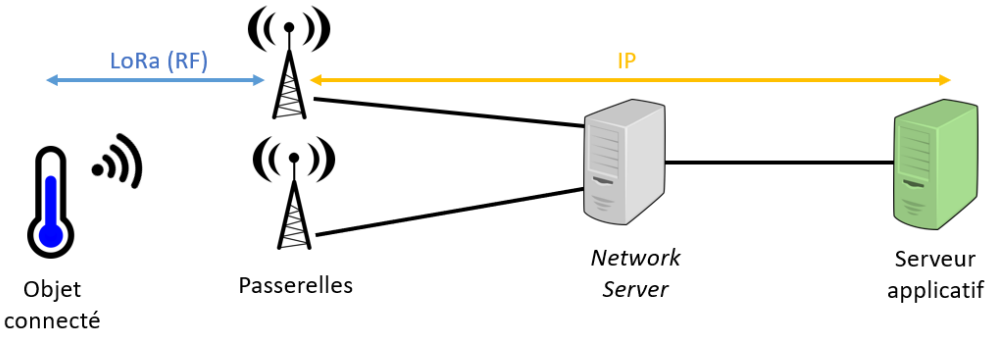

FIGURE 3.14 – Architecture générale d'un réseau LoRaWAN.

14. https://lora-alliance.org/.

Le *Network Server* traite les données en provenance des passerelles. Ce sont des équipements qui font la liaison entre la modulation LoRa, c'est-à-dire la couche physique utilisée par les objets connectés pour communiquer, et le réseau IP. Si les données proviennent d'un nœud légitime enregistré auprès du réseau, elles sont transmises au destinataire, un serveur applicatif le plus souvent.

Le protocole LoRaWAN garantit l'authentification mutuelle entre l'objet connecté et le réseau LoRaWAN, c'est-à-dire le serveur central du réseau, ainsi que l'intégrité et la confidentialité des données de bout en bout [LoR16]. Ces propriétés de sécurité sont assurées par des mécanismes adaptés aux objets contraints pour rester cohérent avec le reste du protocole et sa cible.

Chaque objet équipé d'un module de communication LoRaWAN est personnalisé avec un secret de 128 bits labélisé AppKey et un identifiant unique de 64 bits labelisée DevEUI qui sont stockés dans la mémoire de l'objet. L'authentification mutuelle entre un objet et un réseau est effectuée lors de l'intégration de l'objet au réseau, par la procédure appelée *Over-The-Air Activation* (OTAA). L'authentification repose notamment sur la connaissance partagée du secret AppKey entre l'objet et un serveur d'authentification (non représenté dans la Fig. 24). Suite à une authentification réussie, deux secrets sont dérivés du secret initial :

— un secret labélisé NwkSKey, qui est utilisé pour garantir l'intégrité et la confidentialité des données entre l'objet et le serveur en charge du réseau. Ce secret doit être partagé avec ce dernier.
— un secret labélisé AppSKey, qui est utilisé pour garantir la confidentialité de bout en bout entre l'objet et les serveurs d'applications avec lesquels il communique. Cela permet, notamment, la confidentialité vis-à-vis de l'opérateur du réseau LoRaWAN.

Ces deux secrets et les niveaux de sécurités associés sont similaires à ce qui est proposé par le protocole ZigBee. Les nouveaux secrets sont également stockés sur l'objet de manière pérenne. Cela implique que l'objet doit disposer d'un stockage adapté en fonction des risques auxquels il sera exposé [LoR16]. Toute extraction réussie de ces secrets compromet la sécurité offerte par le protocole.

Pour assurer la sécurité, LoRaWAN s'appuie sur l'algorithme AES du fait de sa réputation validée par des organismes comme le NIST. Les modes d'opérations utilisés par le protocole sont :

— CMAC pour garantir l'authenticité et l'intégrité des données, et
— CTR pour chiffrer les données et garantir leur confidentialité.

Ces choix sont cohérents avec les bonnes pratiques de sécurité en vigueur.

3.3.4 *Wi-Fi*

Le Wi-Fi désigne un ensemble de protocoles réseau sans fil, basé sur les standards IEEE 802.11. Wi-Fi™ est une marque déposée de l'organisation Wi-Fi Alliance™ [15], qui gère les spécifications des protocoles ainsi que la certification des équipements pour valider leur implémentation et garantir leur interopérabilité.

15. https ://www.wi-fi.org/

Le premier standard 802.11 est publié en 1997. De nombreuses versions se sont succédées depuis, qui ont eu pour but d'améliorer les caractéristiques des protocoles comme leur portée ou leur débit de données. Les versions les plus répandues sont le Wi-Fi 4 et le Wi-Fi 5, publiées respectivement en 2009 et en 2014. Nous pouvons noter également le développement de versions dédiées à des cas d'usage particuliers, comme le Wi-Fi HaLow conçu pour l'Internet des Objets en proposant une consommation plus faible et une portée étendue par rapport aux autres versions. Les principaux cas d'usage du Wi-Fi sont la mise en réseau d'équipements à une échelle locale et l'accès à l'Internet de manière générale.

Le Wi-Fi est un protocole bas niveau, situé à la couche 2 du modèle OSI [16]. De ce fait, il gère l'authentification des équipements, l'intégrité et la confidentialité des données à ce niveau uniquement, c'est-à-dire sur le lien radio entre deux équipements. Pour bénéficier de la sécurisation des communications de bout en bout, de l'objet jusqu'au serveur d'applications, de façon similaire à ce que peuvent proposer les protocoles ZigBee ou LoRaWAN, il est nécessaire de faire intervenir d'autres protocoles de plus haut niveau. Toutefois, le Wi-Fi a été conçu pour être interopérable avec son homologue filaire Ethernet et supporte les protocoles de communication standard de l'Internet : IP, TCP/UDP, HTTP et MQTT, par exemple. Il est notamment possible d'utiliser le protocole de communication sécurisé TLS [17] sur un lien Wi-Fi. La figure 3.15 illustre la place du protocole Wi-Fi dans une pile réseau suivant le modèle OSI.

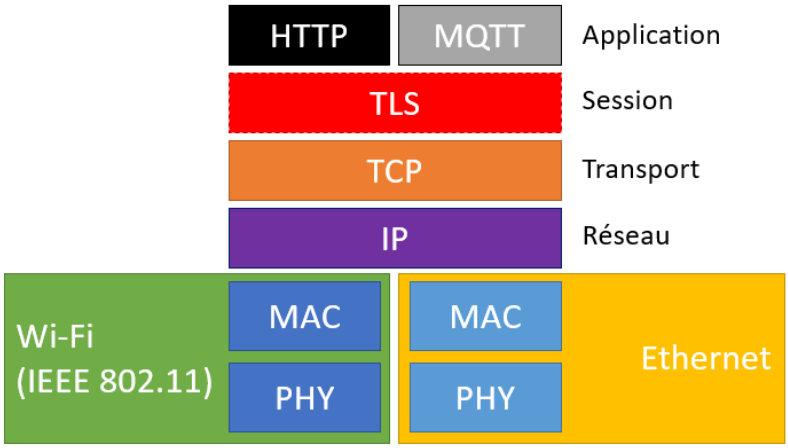

FIGURE 3.15 – Place du protocole Wi-Fi dans une pile réseau suivant le modèle OSI.

Un réseau Wi-Fi est composé de deux types d'équipements :

— Une **station** désigne tout équipement équipé d'un adaptateur sans fil compatible et cherchant à rejoindre un réseau Wi-Fi. C'est typiquement le rôle assumé par un objet connecté domotique nécessitant l'accès à un réseau Wi-Fi pour fonctionner.

— Un **point d'accès** désigne tout équipement diffusant un réseau Wi-Fi et pouvant assurer la passerelle entre ce réseau et un réseau filaire. Il s'agit généralement d'équipements réseaux dédiés, l'exemple le plus commun

16. *Open Systems Interconnection*
17. *Transport Layer Security*

et familier étant le routeur fournit par un Fournisseur d'Accès Internet (FAI).

Deux modes opératoires sont définis par la norme IEEE 802.11 :
— Le mode infrastructure dans lequel les stations se connectent à un point d'accès.
— Le mode ad hoc dans lequel les stations peuvent se connecter les unes aux autres sans point d'accès.

Nous nous concentrons dans la suite sur le mode infrastructure, qui est le plus courant pour les cas d'usage domotique que nous considérons. Par exemple, les objets connectés d'une maison se connectent au réseau Wi-Fi domestique géré par le point d'accès local pour communiquer avec un équipement de supervision, qui peut être ce point d'accès ou bien un équipement y étant relié.

Du point de vue de la sécurité du Wi-Fi, plusieurs standards se sont succédés pour garantir la confidentialité et l'intégrité des données échangées. Les premiers standards, le *Wired Equivalent Protection* (WEP) et le *Wi-Fi Protected Access* (WPA), ont été dépréciés en 2004 suite à la découverte de vulnérabilités dans les algorithmes utilisés [Seb+16], qui affaiblissaient considérablement le niveau de sécurité offert. Les standards actuellement en vigueur sont le WPA 2 et le WPA 3, les successeurs du standard WPA. Le niveau de sécurité de ces standards est considéré comme correct grâce, notamment, à l'utilisation de l'algorithme AES dont la qualité est éprouvée. Comme précédemment, il est important de connaître le standard de sécurité Wi-Fi employé dans une solution pervasive et il est recommandé de choisir le standard le plus récent lors de la conception d'une nouvelle solution.

L'accès à un réseau Wi-Fi repose sur la récupération d'un secret propre au réseau. La méthode d'obtention de ce secret par un nouvel équipement et le niveau d'authentification associé dépend du mode dans lequel est configuré le réseau :
— En mode *Personal*, le secret est géré au niveau du point d'accès. Un nouvel équipement doit connaître le secret pour établir avec succès une liaison avec le point d'accès. La transmission du secret à un nouvel équipement est une opération manuelle souvent réalisée par un utilisateur. Cela peut consister en la saisie d'un mot de passe sur le nouvel équipement. Le mot de passe permet au nouvel équipement d'authentifier le réseau, mais pas nécessairement au réseau d'authentifier le nouvel équipement car le mot de passe est partagé par tous les équipements.
— En mode *Enterprise*, un protocole dédié est utilisé pour authentifier le nouvel équipement avant de lui transmettre le secret. Ce protocole est décrit par la norme IEEE 802.1x et implique un serveur d'authentification distinct du point d'accès. Ainsi, l'authentification du nouvel équipement et la gestion du secret est confiée à ce serveur. Le point d'accès agit comme un relai entre l'équipement et le serveur d'authentification pendant la procédure. L'équipement et le serveur sont capable de s'authentifier mutuellement. Ce mode offre donc un niveau de sécurité accru, mais requiert des infrastructures supplémentaires.

A partir du standard de sécurité *Wi-Fi Protected Access* 2 (WPA2), la confidentialité des données en transit entre un équipement et le point d'accès avec lequel il communique est assurée par l'algorithme AES en mode CTR. La clé

de chiffrement utilisée peut être de 128, 192 ou 256 bits selon la configuration choisie.

Toujours à partir de WPA2, l'authentification et l'intégrité des données en transit sont assurées par l'algorithme AES en mode CBC-MAC.

3.3.5 *Synthèse*

Nous avons synthétisé dans le tableau 3.6 les résultats de notre étude sur les protocoles de communication, en nous concentrant sur les caractéristiques de sécurité principales définies à la section 2.5.

	Niveaux d'authentification	**Moyens d'authentification**	**Confidentialité**	**Intégrité**
ZigBee	Nulle, simple, mutuelle	Secret partagé	Oui	Oui
Bluetooth Low Energy	Nulle, simple, mutuelle	Validation utilisateur	Selon réglage	Selon réglage
LoRaWAN	Mutuelle	Secret partagé	Oui	Oui
Wi-Fi	Simple, mutuelle	Secret partagé	Oui	Oui

TABLE 3.6 – Synthèse de l'étude sur les protocoles sans fil pour objets connectés.

Il ressort de notre étude que les différents protocoles étudiés présentent des **niveaux de sécurités variés**. Par exemple, le Bluetooth Low Energy autorise des échanges non authentifiés, tandis que le LoRaWAN impose une authentification mutuelle entre l'objet et le réseau. Un protocole peut même proposer plusieurs niveaux de sécurité selon la configuration utilisée. Nous pouvons citer de nouveau le Bluetooth Low Energy qui permet, selon configuration, d'échanger des messages en clair ou de manière confidentielle. Gérer cette diversité de configuration représente un défi majeur dans le développement et la sécurisation des environnements pervasifs.

Une autre observation clé est **l'importance de la gestion des secrets** dans les protocoles étudiés. La majorité des protocoles font usage d'un secret ou plusieurs secrets qui sont utilisés pour permettre le bon fonctionnement des fonctionnalités de sécurité, tel que le chiffrement des données pour assurer leur confidentialité. Si ces secrets sont découverts et récupérés par un attaquant, ce dernier est souvent en mesure de compromettre les protections mises en place. Ainsi, la gestion sécurisée de ces secrets est un enjeu majeur.

Dans la prochaine section, nous approfondissons cette notion de gestion des secrets et les problématiques de stockage associées.

3.4 GESTION DES SECRETS ET SÉCURITÉ

La sécurité de nombreux protocoles repose sur la gestion sécurisée de secrets, aussi appelés clés secrètes ou clés cryptographiques, qui sont utilisés par ces protocoles, comme le veut le principe énoncé par Kerckhoffs :

> « la sécurité d'un cryptosystème doit reposer uniquement sur le secret de sa clé. » [Aug83]

Pour stocker de manière pérenne un secret sur un objet connecté, plusieurs méthodes existent. Ces méthodes n'apportent pas le même niveau de sécurité face aux attaques et ont-elles ont des coûts de déploiement variés.

La manière la plus simple de stocker un secret sur un objet connecté est d'enregistrer ce secret dans la mémoire non-volatile de l'objet. La mémoire non-volatile a pour propriété principale de conserver ses informations même lorsque le circuit n'est pas alimenté. Ce type de mémoire permet la persistance d'informations tout au long de la vie de l'objet. Elle est présente sur la majorité des cartes embarquées et des microcontrôleurs qui servent de base aux objets connectés, ce qui en fait un support de stockage très bon marché. Cependant, ce type de mémoire ne possède généralement aucune protection, que ce soit contre les attaques logicielles ou contre les attaques physiques extérieures [Mar+17]. Une mémoire non-volatile ne restreint pas les demandes de lecture faites par la couche logicielle. Si un attaquant parvient à prendre le contrôle de cette couche et à effectuer des lectures de la mémoire à des adresses arbitraires, il est probable qu'il puisse lire le contenu complet de la mémoire. De plus, les mémoires non-volatiles ne possèdent pas de protections contre les attaques physiques. Il est ainsi possible pour un attaquant ayant un accès physique à un objet connecté de récupérer le contenu de sa mémoire non-volatile, puis d'analyser ces données pour en extraire les secrets utilisés par l'objet. Pour limiter ce type d'attaque, des alternatives plus sécurisées ont été développées.

Commençons par les mécanismes de cloisonnement de la mémoire, qui visent à limiter les possibilités d'extraction des attaques logicielles. Avec ce type de mécanisme, une partie de la mémoire non-volatile est isolée de manière matérielle ou logicielle, et le microcontrôleur qui fait fonctionner l'objet connecté alterne entre deux contextes :

— un contexte classique, aussi appelé *Rich Execution Environment* (REE), dans lequel les applications de base s'exécutent et où elles peuvent accéder de manière non restreinte aux périphériques et échanger des données avec le monde extérieur.
— un contexte sécurisé, aussi appelé *Trusted Execution Environment* (TEE), dans lequel les fonctionnalités critiques de sécurité s'exécutent et les interactions avec l'extérieur sont grandement limitées pour prévenir toute fuite d'information. Depuis ce contexte, il est possible d'accéder à la partie isolée de la mémoire.

La transition d'un contexte à un autre s'effectue via une procédure dédiée qui protège autant que possible l'isolation des données propre à chaque contexte. Toute tentative par une application d'accéder à un espace mémoire situé en dehors de son contexte résulte en la levée d'une alerte et l'arrêt de l'application fautive. Cela permet de limiter la portée de l'extraction, idéalement au contexte

classique dans lequel s'exécutent les applications les plus susceptibles d'être compromises. Ce type de protection est implémenté par la majorité des fabricants de microcontrôleurs sous différents noms : TrustZone pour ARM ou SGX pour Intel. Il faut toutefois choisir un microcontrôleur qui dispose de l'option, ce qui implique souvent un surcoût. Ainsi, cette technologie est utilisée dans des équipements manipulant des données sensibles, comme des terminaux bancaires ou des serveurs d'applications. L'avantage de ce type de protection est son intégration native au microcontrôleur, mais elle peut se révéler moins sécurisée que des éléments matériels dédiés, les *Secure Elements* [Glo15], en particulier face aux attaques physiques.

Un *Secure Element*, abrégé SE, est un coprocesseur dédié au stockage sécurisé de secrets, ainsi qu'à la réalisation de calculs et d'opérations cryptographiques telles que la signature de données ou le chiffrement. Il s'agit généralement d'un élément matériel externe au microcontrôleur principal, qui doit donc être intégré lors de la conception de la partie électronique d'un objet connecté. Ce coprocesseur dispose de protections renforcées contre un large panel d'attaques physiques visant à récupérer les secrets d'un objet, telles que l'analyse de la consommation énergétique [KJJ99] ou les attaques en faute [Pap+20]. Grâce à une série de capteurs embarqués, un *Secure Element* est capable de détecter les tentatives d'intrusion physique et de réagir, en conséquence, en supprimant la mémoire interne pour empêcher toute compromission des secrets qui y sont stockés, par exemple. Un *Secure Element* est conçu de manière sécurisée. Il respecte un ensemble de normes, comme les Critères Communs (CC) [18] et peut faire l'objet d'une certification par un organisme de qualification. Ce niveau de sécurité implique cependant un surcoût non négligeable par rapport aux méthodes précédentes. Il y a d'abord le coût d'acquisition du coprocesseur en lui-même et la conception ou la modification du circuit électronique de l'objet connecté pour intégrer ce nouvel élément. Il faut également considérer la formation des développeurs embarqués qui devront concevoir leurs applications en faisant bon usage des fonctionnalités du *Secure Element*. Pour cette raison, le déploiement et l'utilisation de *Secure Element* est généralement réservé à des milieux très sensibles comme le domaine militaire, les applications industrielles sensibles ou la télémédecine.

3.4.1 *Synthèse*

Le tableau 3.7 présente une synthèse des principales méthodes pour stocker des secrets sur les objets connectés que nous avons étudiées.

18. `https://www.commoncriteriaportal.org/`.

	Niveau de protection du secret	**Coût supplémentaire**
Mémoire morte	Nul	Nul
Cloisonnement mémoire REE/TEE	Modéré	Modéré
Secure Element	Fort	Fort

TABLE 3.7 – Synthèse des moyens de stockage pérenne pour un secret.

Selon la méthode de stockage choisie pour conserver les secrets d'un objet connecté, le **niveau de protection** peut grandement varier. En combinant cela avec la diversité des configurations de sécurité existantes dans les protocoles de communication, que nous avons illustré dans la section 3.2, cela illustre parfaitement l'hétérogénéité globale de la sécurité dans l'accès et l'utilisation des objets connectés. Dans ce contexte, si nous souhaitons avoir des solutions pervasives aussi ouvertes que possible qui puissent intégrer des objets connectés avec des niveaux de sécurité variés, il nous paraît essentiel de renseigner les applications pervasives sur l'existence de ces niveaux de sécurité pour permettre leur prise en compte dans le traitement des données issues des objets, ou dans le choix des objets ou services à utiliser.

Nous constatons également un vide dans les solutions de stockage étudiées, entre d'un côté la mémoire morte qui n'offre aucune protection sur le stockage des secrets et de l'autre côté le cloisonnement mémoire ainsi que le *Secure Element*, qui fournissent un niveau de protection plus élevée mais nécessitant l'acquisition de microcontrôleurs spécifiques ou de matériel supplémentaire. Il nous parait donc délicat de déployer ces solutions de sécurisation à grande échelle, en particulier sur des objets connectés cherchant à rester bon marché et visant des cas d'usage peu critiques comme la domotique. Pour ces objets, une solution de stockage offrant un niveau de protection basique avec un coût de déploiement faible serait l'idéal.

Pour tenter de trouver un tel compromis, nous avons étudié une technologie prometteuse qui permet, notamment, le stockage sécurisé de secrets : les fonctions physiques non-clonables.

3.5 FONCTIONS PHYSIQUES NON-CLONABLES

Les fonctions physiques non-clonables, *Physical Unclonable Functions* en anglais et que nous abrégerons en PUF par la suite, sont des systèmes physiques qui réagissent à un stimulus d'entrée en produisant une sortie unique et caractéristique du système utilisé. Présentées pour la première fois par R. Pappu [Pap01] en 2001 sous le nom *Physical one-way functions*, une PUF a pour principe de tirer parti des variations causées par tout procédé de fabrication ou du désordre physique qui affecte la majorité des systèmes physiques [RH14].

D'un point de vue haut niveau, une PUF peut être modélisée comme une fonction prenant une valeur en entrée, le **challenge**, et fournissant une valeur en sortie, la **réponse**, comme illustré par la figure 3.16.

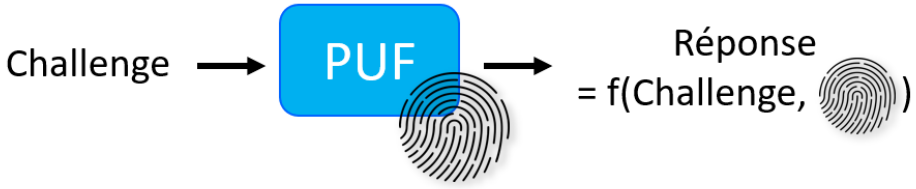

FIGURE 3.16 – Modélisation haut niveau d'une PUF

La réponse dépend non seulement du challenge fourni en entrée, mais également d'une valeur interne à la fonction, son **empreinte**. Cette empreinte est différente pour chaque instance de PUF. En pratique, cette empreinte dépend des caractéristiques physiques du système. Elle n'est pas facilement observable et elle est donc très difficile à reproduire, ce qui rend le clonage d'une instance de PUF donnée très complexe. C'est là une des caractéristiques clés des PUF, qui motive leur utilisation pour sécuriser des systèmes.

Plusieurs cas d'usage pour les PUF dans le domaine de la sécurité font actuellement l'objet de recherches, par exemple :

— la génération d'aléas physiques pour une utilisation comme source aléatoire fiable dans des procédés cryptographiques [HBF09],
— le stockage sécurisé de secrets à faible coût, par rapport aux solutions actuelles présentées dans la section 3.3 [SGP18],
— l'identification et l'authentification d'équipements à faible coût [SD07 ; Mae12]. Par identification, nous entendons la production d'un identifiant de manière fiable et répétable dans le temps. Par authentification, nous entendons l'action de garantir l'identité d'un système, comme nous l'avons présenté dans la section 2.5.3.

Dans la suite de cette section, nous commencerons par présenter quelques exemples concrets de systèmes PUF. Nous présenterons ensuite les principales caractéristiques des PUF à considérer pour faire de l'authentification, puis pour finir le principe général de l'authentification basée sur la technologie PUF.

3.5.1 *Exemples concrets de systèmes PUF*

Diverses architectures de PUF ont été proposées dans la littérature, exploitant des phénomènes physiques variés dans des domaines comme l'optique [Pap01], les champs électromagnétiques ou encore l'électronique [Gas+02]. Nos travaux portant sur la sécurisation d'objets connectés composés principalement de circuits électroniques, nous nous sommes concentrés sur cette famille d'architectures. Les PUF de cette famille, souvent désignée par le terme générique *Silicon PUF*, sont formées à l'aide de composants et de structures classiques dans les circuits intégrés, et peuvent donc facilement être intégrées à des circuits plus importants comme des microcontrôleurs, par exemple.

Les *Silicon PUF* sont divisées en deux catégories principales :

— les ***delay-based* PUF** : ces architectures de PUF sont basées sur le délai de propagation d'un ou plusieurs signaux au travers de circuits et de composants.

— les ***memory-based* PUF** : ces architectures exploitent certaines caractéristiques des circuits utilisées pour le stockage d'informations.

Nous allons présenter deux exemples concrets de PUF, parmi les architectures les plus classiques, pour illustrer le fonctionnement de chacune de ces catégories : l'Arbiter PUF [Gas+02] pour les *delay-based* PUF et la SRAM PUF [HBF09] pour les *memory-based* PUF.

Le principe de l'Arbiter PUF est de propager un signal dans deux pistes de conception identique, comportant k blocs de commutation permettant de modifier le chemin emprunté par les signaux en entée. Au bout des pistes, un composant réagit à l'arrivée des signaux par l'émission d'un bit selon l'ordre d'arrivée. L'architecture générique du circuit est illustrée par la figure 3.17.

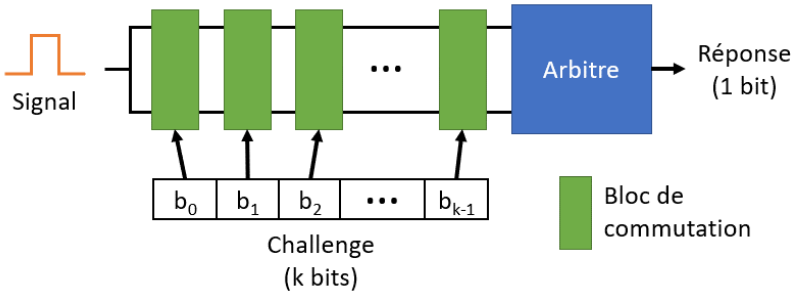

FIGURE 3.17 – Architecture d'une Arbiter PUF.

Les blocs de commutation sont pilotés par un vecteur de k bits, qui est le challenge pour ce type de PUF. Si un circuit seul produit un seul bit de sortie, il est possible d'utiliser plusieurs circuits en parallèle pour obtenir un vecteur réponse de la taille souhaitée. L'Arbiter PUF peut accepter jusqu'à 2^k challenges différents, ce qui en fait une PUF capable de générer de nombreuses réponses, aussi appelée une PUF forte comme nous le verrons dans la section 3.4.2. C'est une caractéristique importante pour réaliser de l'authentification.

La SRAM PUF est une architecture de PUF qui se base sur les circuits mémoire SRAM [19]. Ce type de mémoire est un assemblage de cellules individuelles pouvant stocker un bit d'information. Une cellule est composée de deux circuits inverseurs montés en boucle, ce qui crée deux états stables pour maintenir un signal stocké dans le circuit. Durant le fonctionnement normal de la mémoire, les signaux stockés dans les cellules dépendent des informations stockées dans la mémoire et les opérations d'écriture modifient les signaux enregistrés au besoin. Cependant, au démarrage de la mémoire et si cette dernière n'est pas initialisée, aucune information n'est encore stockée, donc aucun signal n'est imposé aux cellules qui composent la mémoire. La valeur du signal stocké par la cellule va alors dépendre, notamment, du biais de fabrication de la cellule comme l'a constaté D. Holcomb et son équipe [HBF09]. Ils ont montré qu'en lisant la valeur d'une plage mémoire de taille suffisante juste après la mise sous tension de la mémoire, il est possible d'obtenir une empreinte caractéristique, comme présenté par la figure 3.18.

19. *Static Random Access Memory*.

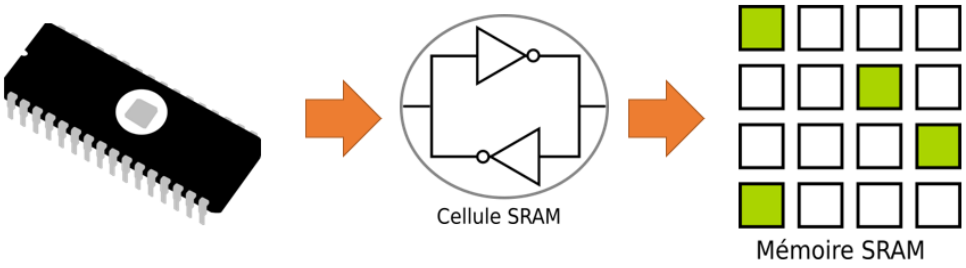

FIGURE 3.18 – Principe de fonctionnement d'une SRAM PUF.

Pour ce type de PUF, le challenge est une adresse mémoire ou une plage d'adresses qui définit la zone mémoire à lire pour générer la réponse. Le nombre de challenges possibles est limité par la taille de la mémoire et de la taille zone nécessaire à la génération d'une réponse. En pratique, le nombre total de challenges possibles est faible par rapport à des modèles comme l'Arbiter PUF. Ainsi, les SRAM PUF sont désignées comme des PUF faibles plutôt que des PUF fortes et elles sont réputées plus adaptées pour du stockage sécurisé de secrets [RH14]. Le principal avantage de la SRAM PUF est la possibilité de tirer parti des mémoires SRAM déjà présentes sur des systèmes embarqués, ce qui réduit grandement le coût d'intégration. Toutefois, la zone mémoire SRAM utilisée comme PUF ne peut être utilisée par une autre application sans perte de l'empreinte.

Dans les sections suivantes, nous adopterons le point de vue d'un utilisateur de système PUF, nos travaux ayant pour vocation d'utiliser les PUF de manière générale pour concevoir des solutions d'authentification légères et sécurisées, sans dépendre d'une architecture de PUF particulière.

3.5.2 *Caractéristiques principales d'une PUF*

Pour comparer les performances de diverses architectures de PUF et leur adéquation pour réaliser de l'authentification, nous avons constitué une liste de caractéristiques clés issues de la littérature [Arm+11] ainsi que les métriques associées.

Pour présenter ces caractéristiques, nous utiliserons les notations suivantes :
— C désigne un challenge pour l'architecture de PUF considérée ; c'est-à-dire un vecteur de bits d'une taille donnée. C_i désigne le challenge d'indice i dans l'espace des challenges possibles.
— $P_A(x)$ représente la fonction de l'instance A de la PUF considérée. Ainsi, $P_A(C_i)$ est la réponse de l'instance A de la PUF au challenge C_i.
— $CRP_{i(A)}$ désigne le couple formé par le challenge Ci et la réponse associée de l'instance A ; c'est-à-dire (C_i, $P_A(C_i)$). De manière générale, CRP est l'abréviation de *Challenge Response Pair*.

Voici maintenant les caractéristiques des PUF :
— la **taille de l'espace de challenges**. Il s'agit du nombre de challenges différents qu'il est possible de soumettre à une architecture de PUF donnée, ce qui donne également le nombre de CRP pouvant être générés. Cette métrique est souvent liée à un ou plusieurs éléments constitutifs de l'architecture considérée. Pour l'Arbiter PUF présentée précédemment,

le nombre de challenges varie avec le nombre de blocs de commutation dont est pourvue la PUF. Cette métrique est utilisée pour classer les architectures en deux catégories principales [RH14] : les PUF faibles et les PUF fortes, respectivement *weak* PUF et *strong* PUF en anglais. Les PUF faibles possèdent un espace de challenge de taille faible par rapport au PUF fortes. Elles sont principalement utilisées dans l'identification de système et le stockage sécurisée, tandis que les PUF fortes sont utilisées pour réaliser des protocoles d'authentification.

— le **taux d'erreur**. Par nature, la plupart des architectures PUF exploitent des instabilités physiques. Il en résulte que l'envoi répété d'un même challenge C_i à une instance de PUF donnée peut générer des réponses légèrement différentes, comme illustré par la figure 3.19.

FIGURE 3.19 – Envoi d'un même challenge sur une PUF donnée pour déterminer le taux d'erreur.

Ainsi, les réponses d'une PUF sont bruitées. Pour évaluer l'intensité de ce bruit, la méthode la plus courante consiste à fixer une réponse de référence puis à calculer le nombre de bits qui diffèrent entre une réponse donnée et la référence, aussi appelée la distance de Hamming [Kim+18]. Le taux d'erreur est obtenu en divisant la distance moyenne constatée par la taille d'une réponse. Pour faire de l'authentification, l'idéal est d'avoir un taux d'erreur aussi faible que possible pour assurer la fiabilité de la procédure. Si le taux d'erreur est trop élevé, il est possible des techniques de correction d'erreurs comme le *fuzzy extractor* [VH+12] pour fiabiliser les réponses de la PUF.

— la **distance inter-PUF**. Il s'agit d'évaluer la diversité des réponses fournies par plusieurs instances d'une architecture de PUF donnée à un challenge donné. Pour cela, un même challenge C_i est envoyé à plusieurs instances de PUF, comme illustré par la figure 3.20.

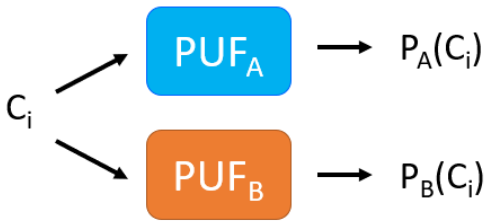

FIGURE 3.20 – Envoi d'un même challenge sur deux instances d'une PUF pour déterminer la distance inter-PUF.

La distance de Hamming est ensuite calculée entre les différentes réponses pour déterminer leur ressemblance. Pour de l'authentifica-

tion, il faut s'assurer que cette distance est suffisamment élevée en moyenne entre les instances pour être capable de les distinguer.

Ces caractéristiques nous permettent de définir des contraintes sur les types de PUF que nous pouvons supporter dans les protocoles d'authentification, que nous allons présenter dans la section suivante.

3.5.3 *Authentification avec une PUF*

L'authentification à l'aide d'une PUF utilise les CRP générés par une instance PUF pour mettre en place une authentification par challenge-réponse. Un des premiers exemples d'un tel protocole d'authentification est présenté par G. E. Suh et S. Devadas [SD07] pour authentifier des circuits intégrés, et, par extension, les objets utilisant ces circuits, en les équipant de circuits PUF. Ce protocole se déroule en deux étapes : une étape d'enregistrement et une étape d'authentification.

Durant l'enregistrement, une identité est créée et assignée au circuit. Puis un certain nombre de challenges choisis aléatoirement sont envoyés à la PUF du circuit et les réponses de la PUF sont collectées. Les CRP ainsi générés sont attachés à l'identité du circuit et sont stockés dans une base de données, appelée registre, gérée par un tiers de confiance. Cette première étape doit être réalisée dans un espace sécurisé pour garantir la confidentialité des CRP, qui seront utilisés à l'étape suivante pour authentifier le circuit. La figure 3.21 résume l'étape d'enregistrement.

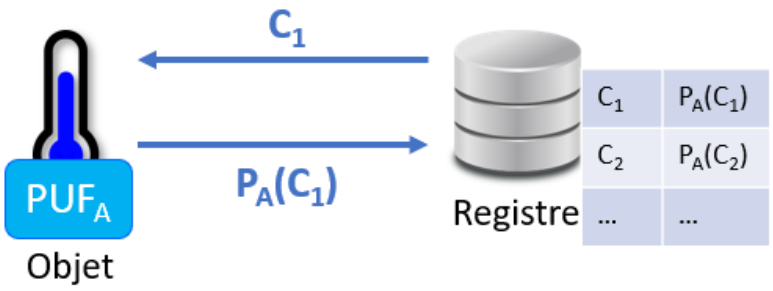

FIGURE 3.21 – Etape d'enregistrement d'un protocole d'authentification PUF.

L'étape d'authentification se déroule une fois l'objet et son circuit déployé sur le terrain, lorsqu'un autre équipement, un serveur de supervision, par exemple, souhaite vérifier l'authenticité de l'objet. Le serveur doit tout d'abord contacter le registre et récupérer un challenge connu pour l'objet concerné. Le serveur soumet ensuite ce challenge à l'objet, qui génère la réponse associée en utilisant son système PUF. L'objet transmet cette réponse au serveur, qui va vérifier la correspondance entre la réponse générée par l'objet et la réponse enregistrée par le registre. A cette étape, il est souvent nécessaire de prendre en compte les erreurs potentielles dans les réponses et, si nécessaire, de les corriger en appliquant des codes correcteurs d'erreurs [VH+12] pour fiabiliser la sortie du circuit PUF. Si, après fiabilisation, la réponse récupérée par le serveur et celle conservée par le registre correspondent, le serveur a authentifié l'objet. Sinon, l'authentification échoue. Cette seconde étape est illustrée par la figure 3.22.

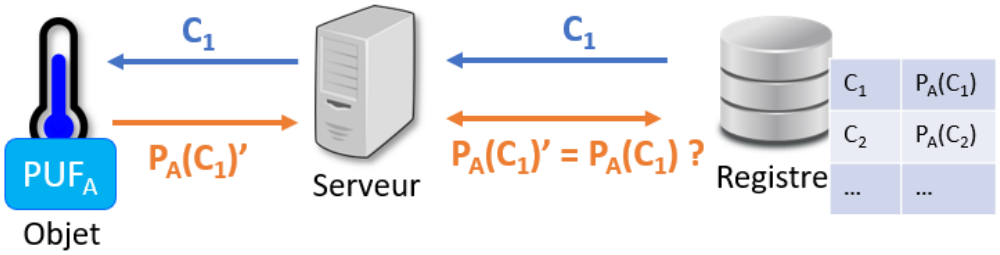

FIGURE 3.22 – Etape d'authentification d'un protocole d'authentification PUF.

Si l'authentification réussie, le CRP utilisé est supprimé du registre pour éviter d'être utilisé de nouveau. Cela signifie qu'il faut avoir suffisamment de CRP, ou données d'authentification, pour couvrir la durée de vie de l'objet.

D'autres protocoles d'authentification pour objets connectés basés sur la technologie PUF ont également été proposés, la recherche étant active dans le domaine. Nous allons présenter quelques-uns de ces protocoles dans la suite de cette section pour identifier leurs points communs et leurs limitations.

La recherche est active dans le domaine de l'authentification légère utilisant la technologie PUF, pour des cibles contraintes comme les objets connectés. Nous avons rencontré durant nos travaux plusieurs propositions de protocoles que nous présentons ici, et dont nous analysons les similarités et les différences avec notre contribution.

Le protocole intitulé T2T-MAP de K. Lounis et M. Zulkerine [LZ21] est un protocole qui permet l'authentification mutuelle entre des objets connectés avec l'aide d'un équipement intermédiaire appelé passerelle. Il est nécessaire que les objets connectés, ainsi que la passerelle, soient équipés d'un circuit PUF pour utiliser le protocole. Les auteurs présentent le concept de CRP étendu, qui utilise les réponses issues des circuits PUF de deux objets. Ces CRP étendus sont utilisés pour réaliser l'authentification, chaque CRP permettant d'authentifier un objet particulier. L'étape d'enregistrement consiste à établir un lien sécurisé entre deux objets connectés pour permettre la génération de tels CRP qui sont ensuite stockés sur les objets. Cette étape d'enregistrement doit être répétée pour chaque couple d'objets qui aura besoin de s'authentifier sur le terrain. Nous supposons que l'enregistrement a lieu juste avant le déploiement d'un objet sur le terrain, ce qui ne couvre pas les premières étapes du cycle de vie de l'objet. De plus, l'ajout d'un nouvel objet connecté à un réseau existant nous semble difficile avec un tel processus d'enregistrement, car cela implique la mise en place d'un lien de communication sécurisé avec des objets déjà déployés. Nous notons que l'étape d'authentification de ce protocole ne supprime aucune donnée d'authentification, ce qui signifie que le protocole n'est pas concerné par l'épuisement des données d'authentification. Les auteurs ont implémenté leur protocole sur des cartes de développement embarquées Arduino Mega 2560 et Arduino DUE, aux caractéristiques proches des objets connectés contraints. Ils ont émulé les circuits PUF à l'aide de la fonction de hachage BLAKE.

F. Farha *et al.* ont présentés un protocole d'authentification léger pour objets connectés utilisant la technologie SRAM PUF [Far+21]. Le protocole utilise une passerelle, située au niveau *fog*, en charge de la gestion et de l'authentification des objets connectés. Le protocole est basé précisément sur la technologie

des SRAM PUF, car il utilise les cellules du circuit mémoire SRAM de l'objet connecté. L'étape d'enregistrement consiste en la génération d'une empreinte avec le circuit SRAM PUF de l'objet, puis la transmission de cette empreinte de manière sécurisée à la passerelle. Nous notons que l'enregistrement n'est pas situé dans le cycle de vie de l'objet connecté. De plus, l'ajout d'un nouvel objet à un réseau déjà formé implique d'effectuer l'enregistrement avec une passerelle déjà déployée. Cela nécessite une attention particulière pour assurer la transmission sécurisée des données d'authentification du nouvel objet, ce qui peut complexifier cette opération d'ajout. La phase d'authentification permet à la passerelle d'authentifier l'objet, mais pas l'inverse. Ainsi, le protocole permet de réaliser une authentification simple uniquement. Cela peut être exploité par un acteur malveillant pour effectuer des attaques de type déni de service sur les objets en se faisant passer pour la passerelle et en générant de nombreux messages. Concernant la gestion des données d'authentification, le protocole ne demande pas de suppression de données et n'est donc pas sujet à l'épuisement des données d'authentification. Les auteurs ont implémenté le protocole sur plusieurs cartes Arduino, en utilisant la mémoire SRAM nativement présenté pour construire la SRAM PUF.

M. Barbareschi *et al.* ont présentés un protocole d'authentification mutuelle utilisant une représentation sous forme de graphes des circuits PUF [BDBM18; Bar+19]. Dans leur modèle, ils considèrent des objets connectés et une passerelle qui est responsable à la fois de l'enregistrement des objets dans un environnement sécurisé et de leur authentification une fois qu'ils sont déployés. Nous notons qu'il n'est mentionné explicitement à quel moment du cycle de vie de l'objet l'enregistrement et l'authentification ont lieu. L'étape d'enregistrement est utilisée pour générer les données d'authentification des objets connectés et stocker ces données sur la passerelle. Ces données peuvent être plus ou moins volumineuses selon la durée de vie et le nombre d'objets à gérer. Le protocole nécessite la suppression des réponses PUF après leur utilisation, pour empêcher les attaques par rejeu. Cependant, l'épuisement des données d'authentification n'est que partiellement évoqué dans leurs travaux. Nous notons que, lors de l'étape d'authentification, les réponses PUF sont échangées en clair sur le canal de communication. Le protocole a été implémenté en utilisant un *Field-Programmable Gate Array* (FPGA), un type de circuit intégré à l'architecture programmable, pour simuler l'objet connecté et son circuit PUF, et un ordinateur de bureau pour la passerelle. Ainsi, l'implémentation matérielle présentée est plus rapide mais plus difficile à mettre à jour qu'une implémentation logicielle sur microcontrôleur. De plus, leur démonstrateur utilise une liaison de données avec débit de 10 Go/s pour relier le FPGA à l'ordinateur en charge de l'authentification, ce qui ne nous semble pas cohérent avec un environnement contraint.

3.6 SYNTHÈSE

Dans ce chapitre, nous avons examiné le traitement de la sécurité à plusieurs points clés d'un environnement pervasif : au niveau des plateformes pervasives, des protocoles de communications et des objets connectés. Nous avons ainsi étudié plusieurs exemples de plateformes pervasives, issues de travaux

académiques et d'entreprises dans le domaine de l'informatique et nous avons comparé la manière dont elles facilitent l'usage des objets connectés et leur gestion sécurisée pour les applications pervasives. Nous avons constaté que les plateformes académiques proposent des services pour faciliter l'utilisation des objets connectés et abstraire leur hétérogénéité, mais traitent peu les sujets concernant la sécurité des objets connectés. Inversement, les plateformes proposées par les entreprises mettent en avant leur gestion de la sécurité, notamment la sécurité des objets connectés, mais exposent plus les applications à l'hétérogénéité des objets connectés. Idéalement, une plateforme pervasive devrait combiner à la fois l'abstraction de l'hétérogénéité des objets connectés et leur gestion sécurisée pour faciliter le développement de solutions pervasives sécurisées. Nous avons également remarqué que, dans les plateformes actuelles, les applications n'avaient accès à aucune information relative au niveau de la sécurité des objets qu'elles utilisaient. Or, en examinant les fonctionnalités de sécurité de protocoles de communication couramment utilisés par les objets connectés et le niveau de protection offert par divers dispositifs de stockage de secrets, nous avons montré l'hétérogénéité des configurations de sécurité possibles et la diversité des niveaux de sécurité qui en découle. Nous pensons que les applications pourraient utiliser des informations comme le niveau de sécurité des objets pour choisir les objets les plus sécurisés ou bien adapter leur confiance en fonction de ce niveau de sécurité.

En dernière partie de ce chapitre, nous avons présenté les fonctions physiques non-clonables, ou PUF, une technologie prometteuse pour la sécurisation de système à faible coût via des applications comme le stockage sécurisé de secrets ou l'authentification. Nous nous sommes particulièrement intéressés à ce dernier point, en présentant les caractéristiques principales d'un système PUF pour réaliser de l'authentification. Pour terminer, nous avons présenté quelques protocoles d'authentification utilisant la technologie PUF et les limites que nous avons identifiés sur ces protocoles, en termes d'intégration du protocole au cycle de vie de l'objet connecté et de gestion de l'épuisement des données d'authentification.

Deuxième partie

AIDE AU DÉVELOPPEMENT SÉCURISÉ DES APPLICATIONS PERVASIVES

4

SOLUTIONS DE SÉCURISATION POUR L'INFORMATIQUE PERVASIVE

Dans la première partie de ce document, nous avons présenté le concept d'informatique pervasive, qui vise à rendre service à des utilisateurs de manière aussi transparente que possible. Nous avons vu que le développement de telles solutions est complexe ; car il repose sur l'intégration de nombreux objets connectés en prenant en compte leur hétérogénéité, leur dynamisme et leur sécurité. Sur ce dernier point, nous avons étudié la sécurité de différents sous-systèmes qui composent une solution pervasive : les plateformes, les protocoles de communication et les dispositifs de stockage de secrets embarqués sur les objets. Nous avons constaté que les moyens de sécurité étaient eux-mêmes hétérogènes et dynamiques, ce qui complexifient l'utilisation sécurisée d'objets connectés par les applications pervasives.

Dans ce chapitre, nous commencerons par rappeler la problématique de cette thèse qui concerne la sécurisation des solutions pervasives. Nous présenterons ensuite notre approche globale pour répondre à cette problématique, basée sur une plateforme pervasive permettant aux applications pervasives d'utiliser de manière simple et sécurisée des objets connectés aux propriétés de sécurité hétérogènes et dynamiques. Pour cela, nous avons conçu une extension pour plateforme pervasive dédiée à la gestion sécurisée des objets connectés. Pour faciliter la sécurisation des objets connectés avec de fortes contraintes en termes de ressources et de budget, nous proposons également un protocole de sécurisation léger basé sur la technologie PUF. Ce protocole est pleinement compatible avec notre extension de sécurité pour plateforme pervasive, ce qui permet son utilisation par les applications pervasives de manière transparente.

4.1 PROBLÉMATIQUE

Suite à notre état de l'art, présenté en première partie, nous avons fait les constations suivantes :

— **de nombreuses technologies de sécurité existent et sont déployées à plusieurs niveaux des solutions connectées**. La sécurité est un domaine vaste, avec de nombreuses technologies disponibles pour sécuriser les systèmes. Les solutions pervasives sont constituées de plusieurs sous-systèmes qui doivent être sécurisés de manière cohérente. Actuellement, il existe peu d'approches transversales qui considèrent la sécurité de l'ensemble du système.

— **les objets connectés manquent d'options de sécurisation adaptées**. Les technologies de sécurisation les plus largement déployées sont conçues pour des systèmes informatiques classiques, composés d'ordinateurs et de serveurs puissants. Elles présentent des problèmes de passage à l'échelle pour un déploiement sur de nombreux objets connectés, et des problèmes d'incompatibilité avec les objets les plus contraints en ressources.

— **l'utilisation des objets connectés est complexe et leur nombre ne cesse de croître**. Pour les applications pervasives, il est nécessaire de prendre en compte l'hétérogénéité des interfaces, ainsi que le dynamisme lié à l'ajout et au retrait d'objets. Mais cela doit se faire de manière transparente pour les utilisateurs, qui ne sont pas formés à l'administration de ces systèmes.

— **de multiples plateformes pervasives ont été développées ces dernières années**. D'abord via des travaux académiques innovants sur la simplification de l'accès aux objets, puis par des entreprises du numérique pour se positionner sur le marché de l'IoT avec des solutions sécurisées. Ces plateformes ont pour but commun de simplifier le développement d'applications reposant sur des objets connectés. Toutefois, ces plateformes fournissent rarement un accès à la fois simple et sécurisé aux objets ainsi qu'une bonne gestion de l'hétérogénéité des technologies de sécurité du domaine.

— **les applications pervasives n'ont pas de visibilité sur la sécurité des objets utilisés**. Actuellement, aucune plateforme ne remonte d'informations sur les technologies de sécurité utilisées par un objet ou un service.

Notre objectif est de faciliter le développement de solutions pervasives sécurisées. Pour cela, nous pensons qu'il est essentiel que les applications pervasives puissent utiliser des objets connectés simplement, de manière sécurisée et en ayant confiance dans la sécurité de ces objets.

Pour atteindre cet objectif, nous proposons une plateforme pervasive qui :

1. **gère la sécurité des objets connectés**, en prenant en compte leurs fonctionnalités hétérogènes et le besoin de passage à l'échelle du domaine ;
2. **masque l'hétérogénéité des objets**, en proposant une interface uniforme, sécurisée et simple d'utilisation pour les applications pervasives ;
3. **renseigne sur le niveau de sécurité des objets**, pour permettre aux applications d'adapter leur confiance.

4.2 APPROCHE GLOBALE

Pour atteindre notre objectif, nous proposons une plateforme pervasive sécurisée qui permet le déploiement d'applications et l'accès à des objets avec différentes technologies de communication et de sécurité, comme illustré par la figure 4.1.

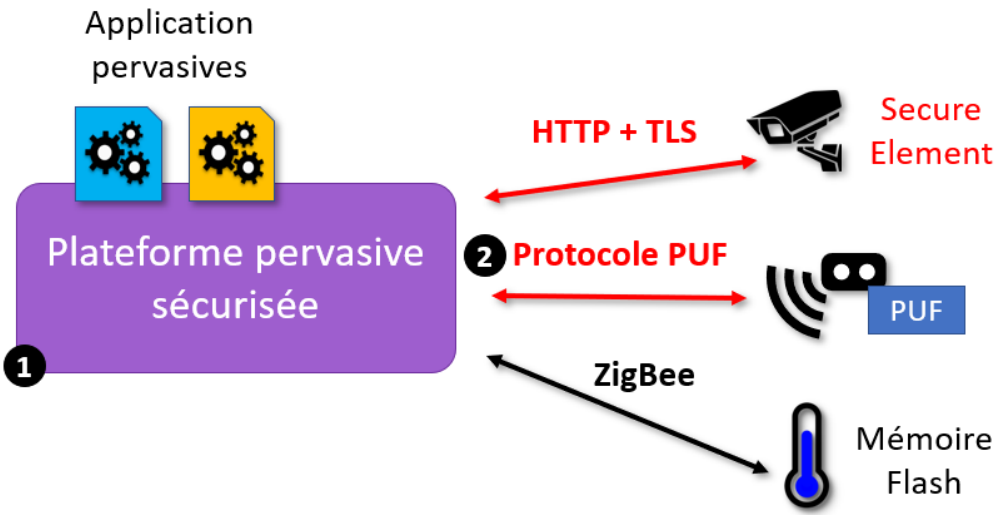

FIGURE 4.1 – Vue globale de notre approche.

La mise en place de notre approche s'appuie sur deux éléments principaux :

— une **extension de plateforme pervasive dédiée à la sécurité (1)**, qui permet de gérer l'enregistrement et la communication sécurisée avec les objets connectés ;

— un **nouveau protocole de communication sécurisé et léger basé sur la technologie PUF (2)**, pour permettre la sécurisation des objets contraints en ressources et en coûts.

Précisément, notre approche s'appuie sur une architecture de plateforme classique, déployée au niveau *fog*, permettant l'accès aux objets connectés de manière abstraite via une approche à services. Notre proposition est d'étendre cette architecture pour gérer la sécurité des objets connectés. Cette extension permet la prise en charge de différents protocoles de communication ainsi que les fonctionnalités de sécurité associées. Elle gère également l'enregistrement des propriétés de sécurité des objets aux niveaux protocolaire et matériel. Ces informations sont utilisées pour déterminer le niveau de sécurité des objets et diffuser cette information aux applications pervasives tout en cachant aux utilisateurs la complexité technologique sous-jacente. Nous avons choisi une plateforme *fog* pour pouvoir prendre en compte des objets contraints pouvant communiquer uniquement sur un réseau local, toutefois une partie significative de cette contribution peut être utilisée pour une plateforme *cloud* également.

Le protocole de communication sécurisé que nous avons conçu a pour but d'être une solution de sécurisation complémentaire dans l'écosystème des objets connectés. C'est une solution qui offre un niveau de sécurité basique avec un coût d'implémentation faible grâce à l'utilisation de la technologie PUF. Cette technologie permet la génération d'une empreinte caractéristique par objet et

la conservation sécurisée de cette empreinte à faible coût par rapport à un secret cryptographique utilisé dans les solutions classiques. Les étapes clés du protocole sont intégrées au cycle de vie de l'objet pour faciliter son déploiement. Enfin, le protocole adopte une approche novatrice dans la gestion des données d'authentification en permettant la génération sécurisée d'une partie de ces données pendant que l'objet est opérationnel. Cela permet une plus grande flexibilité et facilite la prise en charge d'un nombre important d'objets.

Dans les sections suivantes, nous présentons la conception de ces deux contributions de manière détaillée. Puis dans le chapitre suivant, nous montrerons les choix d'implémentation et la validation de ces contributions.

4.3 PLATEFORME PERVASIVE SÉCURISÉE

Nous présentons dans cette section les objectifs spécifiques pour notre extension de sécurité à destination des plateformes pervasives utilisant l'approche à services, puis la conception de l'extension pour atteindre ces objectifs.

4.3.1 *Motivations*

Notre extension doit permettre à une plateforme pervasive :

— **de gérer l'hétérogénéité des configurations de sécurité**, en plus de l'hétérogénéité classique des protocoles de communication.
— **de modéliser la sécurité des objets connectés**, pour connaître les différentes technologies utilisées, qui ne fournissent pas nécessairement les mêmes garanties en termes de confidentialité, d'intégrité et d'authentification.
— **d'informer les applications pervasives sur la sécurité**, avec des données synthétiques et facilement exploitables.

Pour atteindre ces objectifs, nous avons conçu une architecture de plateforme pervasive modulaire dont la vue globale est présentée par la figure 4.2.

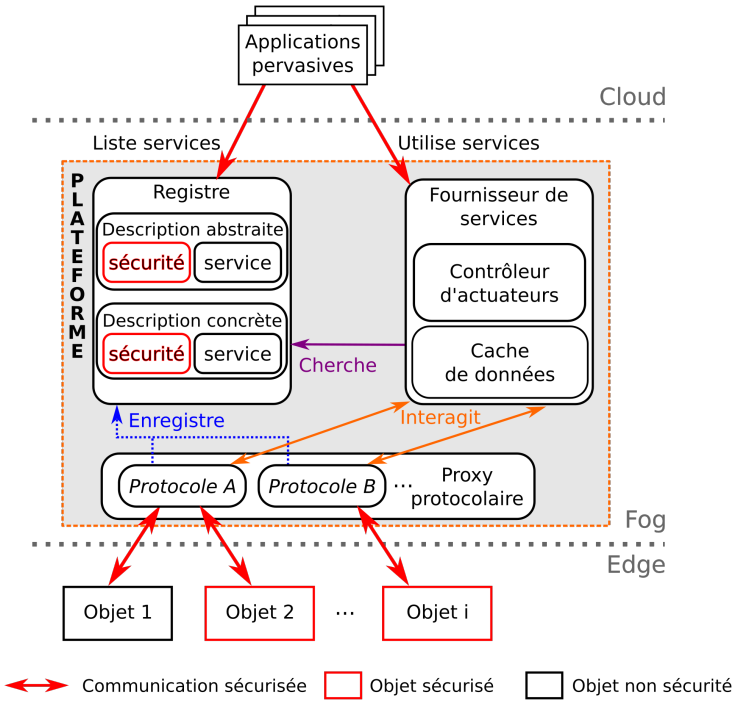

FIGURE 4.2 – Architecture globale de notre plateforme pervasive.

Cette architecture contient trois éléments principaux, que nous appelons modules, qui permettent de faire fonctionner la plateforme. Chaque module a une missions précise :

— le **module Proxy protocolaire** gère les communications entre la plateforme et les objets connectés. Il permet, par exemple, aux objets de s'enregistrer auprès de la plateforme en tant que services et à la plateforme d'envoyer des commandes aux objets.
— le **module Registre** conserve et maintient à jour la liste des objets et des services enregistrés par la plateforme. Les applications peuvent interroger ce module pour obtenir la liste des services qu'elles peuvent utiliser.
— le **module Fournisseur de services** permet aux applications d'accéder à un service donné.

Il s'agit d'une architecture de plateforme classique, similaire à celle de la plateforme iCasa [Lal+14] présentée dans la section 3.1.1. Cette architecture permet un découplage entre les applications pervasives et les objets connectés via une approche à services [Pap03], qui présente les fonctionnalité des objets connectés comme des services abstraits. Cela facilite la gestion de l'hétérogénéité des objets connectés et permet un usage flexibe pour les applications. Nous avons cependant ajouté dans chaque module des fonctionnalités de sécurité pour atteindre les objectifs fixés dans la section précédente. Nous allons présenter ces fonctionnalités dans les sections suivantes.

4.3.2 *Communications sécurisées*

Le **module Proxy protocolaire** a pour objectif premier de permettre d'interagir avec des objets connectés en cachant leur hétérogénéité aux applications pour présenter à la place une interface d'intéraction uniforme. Pour gérer l'hé-

térogénéité des protocoles de communication et de leurs configurations de sécurité, nous avons choisi d'organiser le module en sous-modules spécialisés, plutôt qu'en un seul bloc monolithique. Ainsi, chaque sous-module gère un protocole de communication spécifique avec les différentes configurations de sécurité que supporte ce protocole. Par exemple, notre plateforme possède un sous-module prenant en charge le protocole HTTP(S) avec plusieurs configurations de sécurité, allant de l'échange HTTP, non sécurisé, à l'échange HTTPS avec authentification mutuelle, très sécurisé. Nous avons défini une architecture générale que doivent respecter les sous-modules pour assurer l'interopérabilité avec le reste de la plateforme. Cette architecture est illustrée par la figure 4.3.

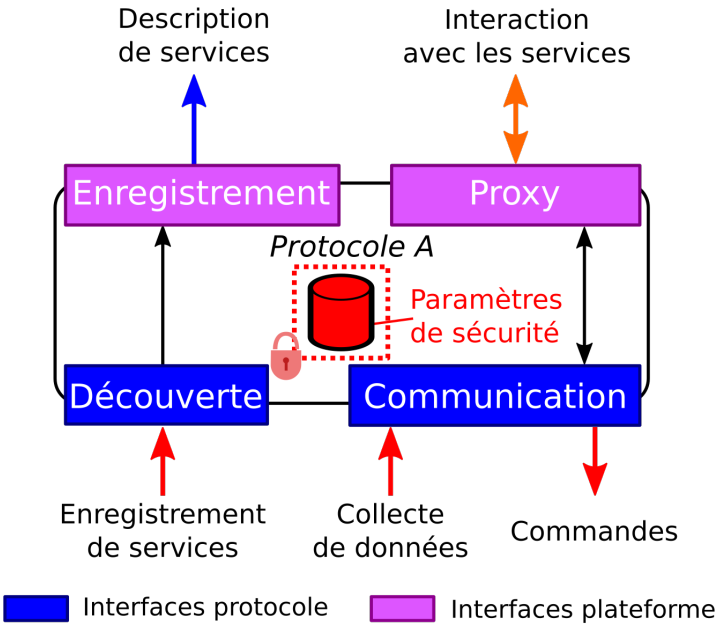

FIGURE 4.3 – Architecture générique d'un sous-module de proxy protocolaire.

Les interfaces sont séparées en deux groupes distincts : les interfaces protocolaires et les interfaces plateforme. Les interfaces protocolaires doivent être adaptées en fonction des caractéristiques et des fonctionnalités du protocole géré par le module. Par exemple, l'interface de découverte peut présenter une fonctionnalité d'auto-détection d'objets si le protocole supporte cette fonctionnalité. Les interfaces plateforme permettent la communication avec le reste de la plateforme et utilisent des structures de données communes à l'ensemble de la plateforme. Enfin, l'architecture prévoit une enclave sécurisée permettant le stockage des configurations du protocole relatives à la sécurité ainsi que le stockage des secrets. Cette enclave doit être en mesure de garantir la confidentialité et l'intégrité des données qui y sont stockées. Chaque sous-module possède sa propre enclave pour permettre le cloisonnement de ces données sensibles et éviter que la compromission d'un sous-module n'affecte le reste des sous-modules.

L'approche par sous-module offre plus de flexibilité et d'évolutivité qu'une approche monolithique. Les sous-modules peuvent être démarrés et arrêtés de manière indépendante, ce qui permet de supporter précisément les protocoles utilisés dans une solution pervasive. Si un nouvel objet doit s'ajouter à

la solution et que cet objet communique avec un nouveau protocole, le sous-module correspondant peut être démarré simplement pour activer le support du nouveau protocole. Concernant l'évolutivité, le support d'un nouveau protocole passe par le développement du sous-module associé. Une fois finalisé, le déploiement du nouveau sous-module est similaire au déploiement d'un sous-module existant. Il n'affecte pas les sous-modules existants et ne perturbe donc pas les opérations de la plateforme, contrairement à une approche monolithique qui demanderait systématiquement la mise à jour de l'ensemble du module et donc l'arrêt temporaire des fonctionnalités de communication de la plateforme avec tous les objets connectés.

En synthèse, l'approche modulaire choisie pour le **Proxy protocolaire** permet à notre plateforme de gérer l'hétérogénéité des protocoles de communication et de la sécurité du domaine des objets connectés tout en proposant une interface d'accès uniforme qui masque cette hétérogénéité, via l'utilisation de sous-modules adaptés et sécurisés.

4.3.3 *Modélisation de la sécurité des objets connectés*

Le **module Registre** permet un couplage faible entre les applications pervasives et les objets connectés en exposant aux applications les fonctionnalités des objets sous forme de services abstraits. Cemodule gère l'annuaire des services enregistrés sur la plateforme et des objets connectés qui fournissent ces services. Les services et les objets sont décrits à l'aide d'un modèle de données que nous avons développé. Ce modèle détaille l'interface fonctionnelle des services, ainsi que les caractéristiques techniques et de sécurité des fournisseurs de services ; c'est-à-dire des objets connectés.

Précisément, le modèle est divisé en deux parties :

— la **description abstraite** d'un service présente des informations de haut niveau sur ce service et sa sécurité. Ces informations sont utiles à la sélection d'un service.

— la **description concrète** d'un service contient les détails techniques liés au fournisseur du service et des informations sur ses moyens de sécurité, comme l'adresse réseau de l'objet proposant le service ou bien les moyens d'authentification supportés. Ces informations sont utiles pour accéder de manière sécurisée au service.

Ce modèle permet une séparation des préoccupations entre la sélection des services réalisée par les applications pervasives en utilisant la description abstraite et le contact sécurisé avec les services réalisé par la plateforme en utilisant la description concrète. Une telle séparation des préoccupations existe déjà dans les technologies orientées service [CL08] telles que les Web Services [W3C04] qui proposent une description de services avec le WSDL [W3C07b] et une description de sécurité avec les profils WS-Policy [W3C07c]. Toutefois, ces descriptions étaient peu utilisées en pratique ; car complexes à mettre en œuvre : elles devaient être rédigées manuellement par les fournisseurs de services, puis diffusées aux consommateurs de services pour que ces derniers puissent utiliser les moyens de sécurité attendus.

Notre approche s'inspire des modèles utilisés dans les technologies orientées services, avec la séparation entre la description du service et la description de la sécurité du service comme l'illustre la vue globale de notre modèle présenté par la figure 4.4.

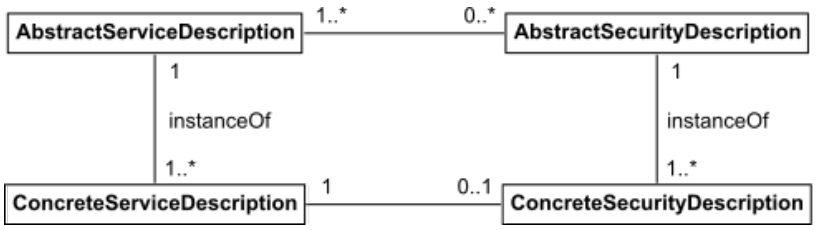

FIGURE 4.4 – Vue d'ensemble du modèle de données.

Cependant, par rapport aux anciens modèles, notre modèle est plus simple d'utilisation pour les fournisseurs et les consommateurs de services, grâce à son intégration dans notre plateforme pervasive. De plus, il représente plus précisément les technologies de sécurité déployées.

Concernant la facilité d'utilisation pour les fournisseurs, ces derniers n'ont plus l'obligation de créer manuellement une description complète de sécurité pour leurs services. A la place, nous nous appuyons sur le **module Proxy protocolaire**, présenté dans la section précédente, pour détecter la configuration de sécurité d'un objet lors de son enregistrement. Chaque sous-module étant spécialisé dans un protocole de communication et les configurations de sécurité associées, il est capable de collecter la majorité des informations nécessaires à la création d'une description concrète de la sécurité de l'objet, représentée par la classe *ConcreteSecurityDescription*. Pour les consommateurs de services, c'est-à-dire les applications pervasives, il suffit de gérer la connexion sécurisée aux interfaces de la plateforme. C'est ensuite cette dernière qui prend en charge la connexion sécurisée aux fournisseurs avec le protocole et la configuration adaptée.

La partie de notre modèle dédiée à la sécurité concrète permet de décrire à la fois la sécurité protocolaire et la sécurité matérielle, en déclinant cette sécurité selon les caractéristiques de **confidentialité**, d'**intégrité** et d'**authentification** que nous avons défini dans la section 2.5. La figure 4.5 donne une vue d'ensemble de cette partie du modèle.

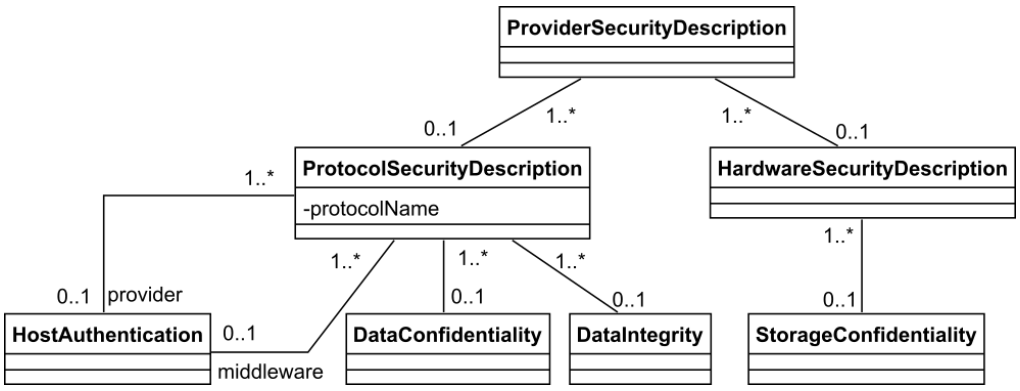

FIGURE 4.5 – Vue d'ensemble de la partie sécurité du modèle - description concrète.

La classe *ProtocolSecurityDescription* décrit les caractéristiques de sécurité du protocole de communication et la classe *HardwareSecurityDescription* décrit les caractéristiques de sécurité propre à l'objet connecté. Ces caractéristiques sont reliées à l'utilisation de primitives de sécurité telles que le chiffrement ou la signature de données, et ces primitives sont implémentées par des algorithmes sécurisés, par exemple. Notre modèle permet de représenter ces relations et d'enregistrer les algorithmes et les secrets utilisés par un objet donné.

Par exemple, la figure 4.6 détaille la description de l'authentification, représentée par la classe *HostAuthentication*.

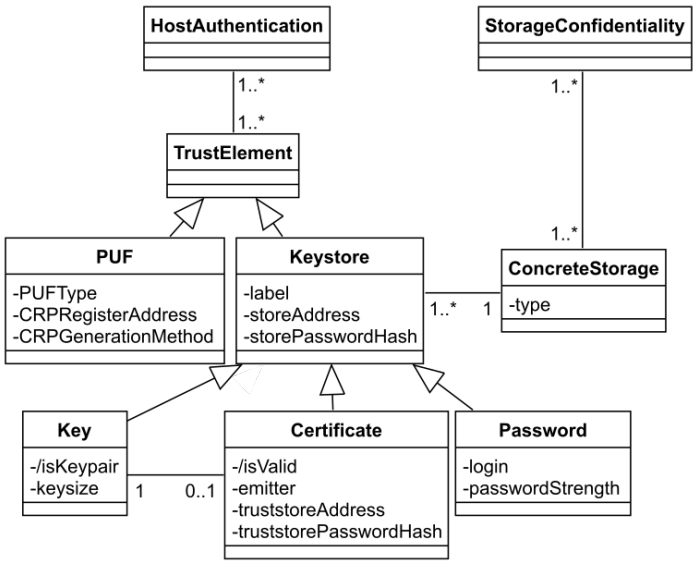

FIGURE 4.6 – Partie authentification du modèle.

Nous considérons que l'authentification repose sur l'utilisation d'un ou plusieurs éléments de confiance, représentés par la classe *TrustElement*. Nous supportons plusieurs moyens d'authentification classiques, tels que l'authentification par mot de passe, représentée par la classe *Password*, et l'authentification par certificat avec une Infrastructure à Clés Publiques (PKI), représentée par la classe *Certificate*. Ces moyens d'authentification reposent sur un secret qui doit être conservé, généralement par les équipements impliqués dans l'authentification. Nous pensons qu'il est important de considérer ce lien entre l'authentification et le stockage, c'est pourquoi nous l'avons matérialisé dans notre modèle en liant la classe mère *Keystore* à la classe *ConcreteStorage*. Ainsi, nous pouvons prendre en compte la manière dont est stocké le ou les secrets utilisés pendant un processus d'authentification. Cette information peut être utilisée pour évaluer le niveau de sécurité du fournisseur, comme nous le verrons dans la section 4.3.4. Nous supportons aussi de nouveaux moyens d'authentification, comme l'authentification utilisant la technologie PUF, en lien avec nos recherches sur le sujet et le protocole que nous présenterons dans la section 4.4. Ce moyen d'authentification est décrit par la classe *PUF*.

Ainsi, notre modèle permet de décrire précisément des configurations hétérogènes de technologies de sécurité utilisées par les objets connectés, à la fois au niveau protocolaire et au niveau matériel.

4.3.4 *Accès aux informations de sécurité pour les applications*

Le modèle présenté précédemment pour décrire concrètement la sécurité des objets connectés nécessite un niveau d'expertise avancé en sécurité protocolaire et matérielle pour être pleinement exploité. Pour rendre les informations de cette partie plus abordables et simplifier leur utilisation, nous avons conçu une description abstraite de sécurité qui synthétise les points clés de la description concrète, présentée dans la section précédente. Le modèle de cette description abstraite est présenté par la figure 4.7.

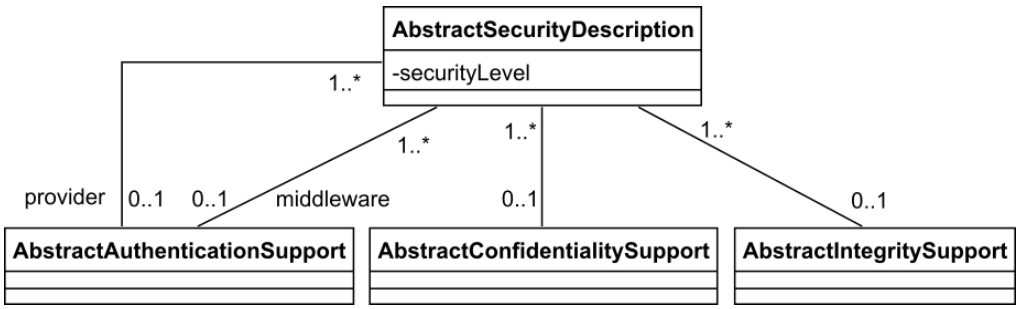

FIGURE 4.7 – Partie sécurité du modèle - description abstraite.

Tout d'abord, les classes *AbstractAuthenticationSupport*, *AbstractConfidentialitySupport* et *AbstractIntegritySupport* renseignent sur la présence ou non de technologies de sécurité assurant les caractéristiques d'authentification, de confidentialité et d'intégrité. Cela permet aux applications pervasives, qui ont accès à cette description abstraite, d'effectuer un tri rapide des services selon leurs caractéristiques de sécurité. Toutefois, ce système ne permet pas de connaître la qualité des technologies de sécurité déployées. Pour cela, nous avons muni la classe *AbstractSecurityDescription* d'un attribut *securityLevel*. Il s'agit d'un score de confiance dans la sécurité décrite par la description concrète de sécurité d'un fournisseur. Le score varie de 0 à 100, 100 indiquant une confiance totale dans la sécurité du fournisseur. Pour calculer ce score, nous avons conçu un algorithme simple qui évalue les technologies de sécurité en nous basant sur les directives de l'ANSSI [ANS20] et du NIST [Bar20]. Nous avons toutefois pris en compte dans notre évaluation le stockage des secrets, en dégradant le score si, par exemple, le secret d'une méthode d'authentification robuste est stocké dans une mémoire morte non protégée. Nous sommes conscients qu'il est possible de concevoir des algorithmes plus complexes pour calculer ce calcul du score de confiance. Nous avons même envisagé la possibilité d'externaliser le calcul de ce score à une entité externe, spécialisée dans la sécurité des objets connectés, qui pourrait mettre à jour régulièrement cet algorithme en fonction des nouveaux protocoles et de la découverte de vulnérabilités, par exemple. Nous n'avons toutefois pas pu explorer pleinement cette approche, cela sortant du périmètre de nos travaux.

En résumé, la description abstraite de sécurité de notre modèle permet aux applications pervasives d'avoir accès à des informations synthétiques et facilement exploitables sur les caractéristiques et le niveau de sécurité des services, pour les utiliser en toute confiance.

4.3.5 *Synthèse*

Pour synthétiser, notre extension de sécurité pour les plateformes pervasives utilisant une approche à services apporte :

— **la gestion de la sécurité hétérogène des objets connectés** avec une approche modulaire pour gérer l'évolutivité et le passage à l'échelle.
— **une description concrète et détaillée des technologies de sécurité** utilisées par les objets connectés, au niveau du protocole de communication et du stockage matériel des secrets.
— **une description abstraite et synthétique de la sécurité**, associée à chaque service, pour faciliter la sélection des services appropriés aux besoins des applications pervasives.

Par la suite, nous nous sommes intéressés à la sécurisation des communications avec les objets connectés contraints.

4.4 PROTOCOLE DE COMMUNICATION SÉCURISÉ ET LÉGER

Dans cette section, nous présentons notre protocole de communication sécurisé et léger, basé sur la technologie PUF.

4.4.1 *Motivations*

Le protocole sécurisé basé sur la technologie PUF doit :

— **permettre une authentification mutuelle** entre un objet connecté et un équipement de supervision ;
— **offrir un compromis entre un niveau de sécurité correct et un coût de déploiement abordable**, pour sécuriser des objets qui ne peuvent pas déployer d'autres solutions ;
— **être intégré au cycle de vie de l'objet**, pour faciliter son déploiement ;
— **supporter le passage à l'échelle**, pour sécuriser des solutions pervasives utilisant de nombreux objets connectés.

Pour atteindre ces objectifs, nous avons conçu, dans un premier temps, un nouveau protocole d'authentification utilisant les fonctions physiques non-clonables, ou PUF, et leur fonctionnalité de génération d'une empreinte propre à un circuit. Une vue d'ensemble du protocole est présentée par la figure 4.8, intégrée dans le cycle de vie d'un objet connecté.

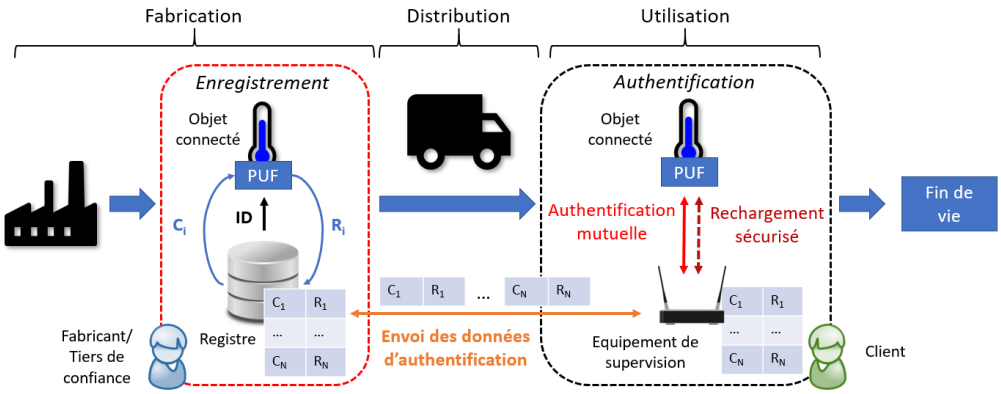

FIGURE 4.8 – Vue d'ensemble du protocole d'authentification PUF dans le cycle de vie d'un objet connecté.

Commençons par définir les trois principaux équipements impliqués dans le protocole :

— un **objet connecté**, fabriqué par un industriel puis distribué et acquis par un client, qui va le déployer dans une solution pervasive. Nous supposons que chaque objet est muni d'un circuit PUF dès sa fabrication, pour permettre de l'authentifier.

— un **registre d'authentification**, qui gère le stockage et la distribution des données d'authentification liés aux objets connectés.

— un **équipement de supervision**, ou passerelle, qui fait partie de la solution pervasive du client, et dont la mission principale est de gérer les objets connectés. Dans notre cas d'usage, c'est cet équipement qui est en charge de l'authentification des objets connectés.

Le protocole comporte trois étapes, dont deux sont classiques et communes à de nombreux protocoles d'authentification basés sur la technologie PUF [SD07 ; BDBM18 ; Ays+15 ; Far+21 ; LZ21]. La dernière étape est propre à notre protocole. Ces étapes sont les suivantes :

— **enregistrement** : il s'agit d'une étape d'initialisation, qui est réalisée juste après la fabrication de l'objet. Le but de cette étape est de collecter un ensemble initial de données d'authentification auprès du circuit PUF. Une fois cette étape réalisée, l'objet est prêt à être vendu et déployé dans une solution connectée.

— **authentification** : une fois que l'objet est déployé sur le terrain et qu'un équipement de supervision va interagir avec lui, pour le paramétrer ou pour collecter des données par exemple, cette étape permet à l'équipement d'authentifier l'objet et inversement. Cette étape s'appuie sur les données d'authentification collectées lors de l'**enregistrement** et récupérées de manière sécurisée lors de l'acquisition de l'objet. Chaque authentification consomme des données qui ne sont pas réutilisables.

— **rechargement sécurisé** : cette étape se déroule également lorsque l'objet est déployé. Elle se déclenche automatiquement lorsque le nombre d'authentifications restantes descend en dessous d'un certain seuil. L'objet et l'équipement de supervision effectuent une authentification mutuelle, puis établissent un canal de communication sécurisé. Ce canal permet à l'équipement de supervision de collecter un nouvel ensemble de don-

nées d'authentification auprès de l'objet sans nécessité d'interrompre sa mission de terrain.

4.4.2 *Protocole d'authentification PUF*

Pour présenter ces étapes dans le détail, nous allons utiliser les conventions suivantes :

- $x||y$ représente la concaténation des valeurs de x et de y ;
- $x \oplus y$ représente l'opération logique OU exclusif bit à bit (ou XOR) des valeurs x et y ;
- $P(x)$ représente la réponse du circuit PUF au challenge x. Nous supposons que cette réponse est fiable, c'est-à-dire que le circuit PUF intègre des mécanismes de correction des erreurs de la réponse, tels que des codes correcteurs d'erreurs ;
- $H(x)$ représente l'empreinte de la valeur x issue d'une fonction de hachage sécurisée ;
- $C(x; K = y)$ représente la sortie d'une fonction de chiffrement symétrique exécutée sur la valeur x avec la clé y ;
- RM_T, PM_T et OM_T représentent respectivement les messages envoyés par le registre (R), la passerelle (P) et l'objet (O), avec T indiquant le type de message.

Nous commençons par présenter les étapes d'enregistrement et d'authentification qui permettent d'aboutir à l'authentification d'un objet équipé d'un circuit PUF, puis nous présentons l'étape de rechargement sécurisé.

4.4.2.1 *Enregistrement*

L'étape d'enregistrement permet d'établir les fondations de la confiance dans l'authenticité de l'objet. Cette étape concerne uniquement l'objet connecté et le registre d'authentification. Elle doit être réalisée dans un environnement sécurisé, dès que possible après la fabrication de l'objet. Cela permet d'être certain que l'objet n'a pas été altéré avant son enregistrement.

Le registre d'authentification va instancier pendant cette étape une structure de données, nommée la table d'authentification, pour stocker les données d'authentification de l'objet. Cette structure stocke une collection de triplets (challenge, réponse, état). Les deux premiers éléments du triplet définissent un *challenge-response pair* (CRP) ; c'est-à-dire un challenge et la réponse du circuit PUF à ce challenge, tandis que le dernier élément indique l'état de l'entrée vis-à-vis du processus d'authentification. Une entrée commence avec un état *valid*, indiquant qu'elle peut être utilisée dans le processus d'authentification. Cet état est mis à jour lorsque l'entrée est utilisée, selon le déroulement du processus d'authentification. Cela est précisé dans la section dédiée à l'authentification. La table d'authentification permet de garder une trace de la façon dont sont utilisées les données d'authentification.

Chaque objet enregistré possède sa propre table. Pour cela, le registre commence par assigner à l'objet un identifiant unique ID_{objet}. Cet identifiant peut être généré par la PUF de l'objet, via une réponse à un certain challenge par exemple. Le registre initialise ensuite une nouvelle table d'authentification as-

sociée à cet identifiant. Ensuite, la génération de CRP commence en suivant la procédure illustrée par la figure 4.9.

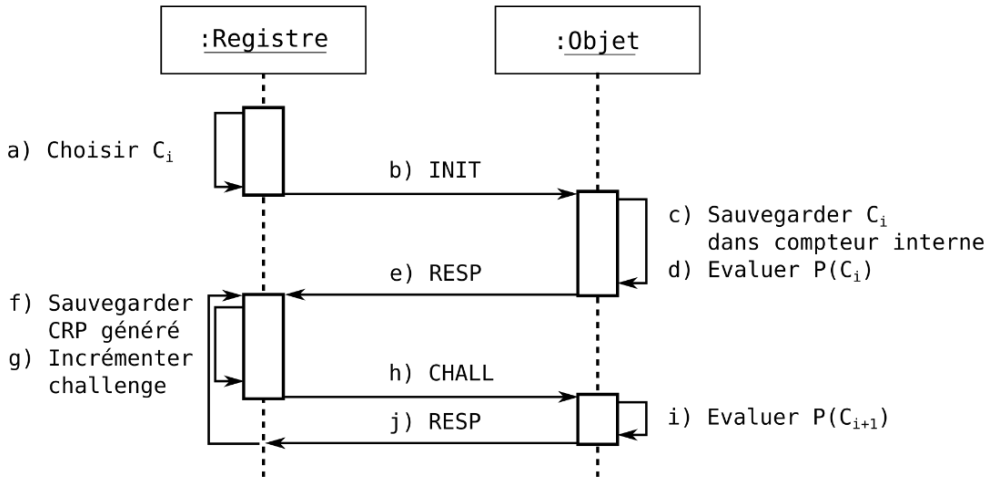

FIGURE 4.9 – Procédure d'enregistrement du protocole.

Le registre d'authentification choisit un entier positif C_i (a), le challenge initial. Cet entier sera également manipulé par l'objet, il faut donc prendre en considération la façon dont ce dernier traite les entiers. Si $I_{\max}$ représente le plus grand entier positif que l'objet puisse traiter, il est recommandé de choisir C_i dans intervalle $\{0; I_{\max} * 0.1\}$. Le choix d'une valeur C_i trop importante pourra limiter le nombre d'authentifications réalisables. L'entier C_i est ensuite envoyé à l'objet en utilisant le message INIT définit par :

$$RM_{INIT} = INIT||C_i$$

L'objet, à la réception de ce message, utilise cet entier C_i pour initialiser un compteur interne (b) . Ce compteur est utilisé pour prévenir des attaques par rejeu et doit être stocké de manière persistante par l'objet. L'objet utilise également C_i comme challenge pour son circuit PUF (c). Le réponse $P(C_i)$ générée par la PUF est renvoyée au registre dans un message RESP (e) défini par :

$$OM_{RESP} = RESP||P(C_i)$$

A la réception de ce message, le registre stocke le CRP ainsi généré $(C_i, P(C_i))$ dans la table d'authentification de l'objet (f). Pour générer le prochain CRP, le registre incrémente la valeur de l'entier pour obtenir C_{i+1} (g) et transmet cette valeur en utilisant un message CHALL (h), dont la définition est similaire au message INIT. Quand l'objet reçoit le message CHALL, il utilise l'entier du message pour générer une nouvelle réponse PUF et la transmet avec un message RESP. Le compteur n'est pas modifié lors de cette opération. Le registre peut ainsi générer le nombre de CRP qu'il juge adéquat en incrémentant la valeur du challenge et en envoyant le message CHALL correspondant.

Pour terminer l'étape d'enregistrement, le registre envoie un message END à l'objet. Cela a pour effet de restreindre le traitement des messages INIT et

CHALL pour empêcher une génération non autorisée de CRP qui révélerait les données d'authentification.

4.4.2.2 *Authentification*

Un fois l'étape d'enregistrement réalisée avec succès, l'objet peut être distribué puis intégré dans un environnement donné, pour faire partie d'une solution applicative pervasive par exemple. Une fois déployé, en accord avec nos hypothèses, l'objet communique avec une passerelle de supervision. L'étape d'authentification se déroule entre l'objet et la passerelle, avec le support du registre d'authentification. L'objectif de cette étape est l'authentification mutuelle entre la passerelle et l'objet. Nous supposons que la passerelle est capable de communiquer avec le registre d'authentification, de l'authentifier et de communiquer de manière sécurisée avec lui.

Pour commencer la procédure d'authentification, la passerelle envoie à l'objet un message ID_REQ. L'objet répond avec un message ID_ANS défini par :

$$PM_{ID\_ANS} = ID\_ANS||ID_{objet}$$

Avec l'identifiant de l'objet, la passerelle peut interroger le registre d'authentification pour récupérer la table d'authentification de l'objet si ce dernier a été correctement enregistré. Il s'agit de la transmission de données d'authentification, illustrée dans la figure 4.8. Dans le cas d'une implémentation réelle, il faudra que la passerelle fournisse des informations supplémentaires pour permettre au registre de s'assurer que les données sont bien transmises à un acquéreur légitime et pas une à personne malicieuse. Nous ne couvrirons pas ici ces mesures qui peuvent s'appuyer sur des mécanismes de sécurisation classiques telles que l'authentification via un certificat délivrée par une Autorité de Certification, la passerelle et le registre étant supposés non contraints. Une fois la passerelle en possession de la table d'authentification de l'objet, elle suit le protocole illustré par la figure 4.10.

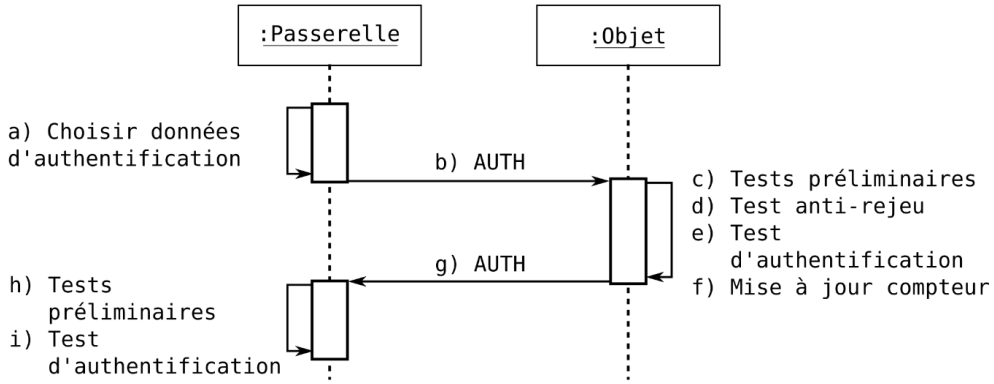

FIGURE 4.10 – Procédure d'authentification mutuelle entre l'objet et la passerelle.

La passerelle choisit dans la table d'authentification deux entrées $(C_n; P(C_n))$ et $(C_{n+1};\ P(C_{n+1}))$, les deux premières entrées valides de la table, par exemple.

Avec ces données, la passerelle calcule son message d'authentification M, défini par :

$$M = ID_{objet}||C_n||P(C_n)\ XOR\ P(C_{n+1})$$

Les deux réponses PUF sont combinées via une opération XOR pour ne pas qu'elles soient transmises en clair, dans le but d'empêcher les attaques par modélisation contre le circuit PUF [Rüh+10 ; RS14] de l'objet. Ce point est discuté plus en détails dans la section 5.2.3.1. M est envoyé à l'objet via un message AUTH défini par :

$$PM_{AUTH} = AUTH||M||H(M)$$

Dans le même temps, la passerelle sauvegarde l'identifiant de l'objet pour mémoriser les objets en cours d'authentification. La passerelle doit également définir un temps maximal pour compléter l'authentification. Si l'objet n'a pas répondu dans le délai imparti, la passerelle considère l'authentification comme échouée et met à jour à l'état des entrées C_n à C_{n+3} de *valid* à *timed-out*. Pour la prochaine authentification, la passerelle utilisera les entrées C_{n+4} et C_{n+5}. Cette mesure évite les problèmes de désynchronisation qui pourraient subvenir si l'objet reçoit le message de la passerelle mais la réponse de l'objet est perdue, pour des raisons accidentelles ou malicieuses. Pour prévenir l'épuisement rapide des données d'authentification, le nombres d'entrées avec un état *timed-out* devrait être surveillé. Si un objet génère un nombre important de mises à jour de ce type sur un intervalle de temps donné, cela peut être le signe d'un défaut réseau ou d'une attaque. Par conséquent, une alarme devrait être levée et l'objet en question isolé, c'-est-à-dire temporairement ne plus initier d'authentification avec cet objet et ne pas accepter de requêtes de sa part. L'isolation peut être levée après que la situation ait été clarifiée par des opérateurs qualifiés.

Lorsque l'objet reçoit le message de la passerelle, il commence par effectuer des tests préliminaires (c) qui consistent en une vérification de l'intégrité du message grâce à son empreinte et une vérification de l'identifiant pour valider que le message est bien destiné à l'objet. Si ces tests réussissent, un test anti-rejeu (d) est ensuite effectué, qui compare le challenge reçu C_n au challenge sauvegardé dans le compteur de l'objet. Le test est validé si C_n est supérieur ou égal au challenge sauvegardé. Si un de ces tests échoue, le traitement du message prend fin sans notification à la passerelle. Si le message est valide, l'objet utilise la valeur C_n et son circuit PUF pour générer les valeurs $P(C_n)$ et $P(C_{n+1})$, pour calculer $P(C_n) \oplus P(C_{n+1})$ et pour comparer le résultat à la valeur reçue dans le message. Si les valeurs correspondent, la passerelle est authentifiée du point de vue de l'objet et la valeur du challenge C_{n+4} est chargée dans le compteur (f) pour éviter une attaque par rejeu du message.

L'objet calcule ensuite son propre message d'authentification N et l'envoie à la passerelle avec un message AUTH (g). Ces messages sont définis par :

$$N = ID_{objet}||P(C_{n+2}) \oplus P(C_{n+3})$$

$$OM_{AUTH} = AUTH\,||N||\,H(N)$$

En supposant que le message est réceptionné par la passerelle dans le délai prévu pour l'authentification, cette dernière vérifie son intégrité avec l'empreinte $H(N)$, puis contrôle si l'identifiant ID_{objet} correspond à un objet actuellement en cours d'authentification (h). Si ces tests sont validés, la passerelle récupère dans la table d'authentification de l'objet les entrées C_{n+2} et C_{n+3} puis calcule la valeur $P(C_{n+2}) \oplus P(C_{n+3})$. Cette valeur est comparée à la valeur reçue dans le message (i). Si les valeurs correspondent, l'objet est authentifié du point de vue de la passerelle et l'état des entrées C_n à C_{n+3} est mis à jour de *valid* à *success*. Dans le cas contraire, l'authentification échoue et l'état des entrées C_n à C_{n+3} est mis à jour de *valid* à *fail*. En effet, quel que soit l'issue de l'authentification le compteur de l'objet ne permet pas la réutilisation de ces entrées.

A la fin de la procédure d'authentification, si toutes les étapes ont été validées, l'objet et la passerelle sont mutuellement authentifiés et quatre CRP ont été utilisés. Il est possible de répéter cette procédure autant de fois que nécessaire, tant que le nombre d'entrées valides de la table d'authentification est suffisant.

4.4.2.3 *Rechargement sécurisé*

Nous avons vu que la procédure d'authentification utilisait des CRP de manière non réutilisable et causait donc l'épuisement progressif des données d'authentification d'un objet. Si la table d'authentification d'un objet venait à se vider totalement, il ne serait plus possible de procéder à des authentifications avec cet objet. Pour éviter cette situation, notre protocole possède une $3^{\text{ème}}$ étape qui permet le rechargement de la table d'authentification d'un objet de manière sécurisée tout en laissant l'objet en place dans son environnement sans affecter le statut de déploiement de l'objet. De plus, cette étape de rechargement sécurisé requiert uniquement un CRP pour être effectuée.

Pour commencer, la passerelle choisit dans la table d'authentification une entrée valide C_n. Avec ces données, elle prépare son message d'authentification M', défini par :

$$M^{'} = ID_{objet}\,||C_n||\,C\,(P\,(C_n)\,;K = P\,(C_n))$$

Encore une fois, la réponse PUF $P(C_n)$ n'est pas transmise en clair pour éviter les attaques par modélisation. Ainsi, seul un équipement ayant accès à la table d'authentification, la passerelle, ou ayant accès au circuit PUF ayant généré cette réponse à l'origine, l'objet, peut retrouver cette valeur. M' est transmis à l'objet via un message REFILL_AUTH défini par :

$$PM_{REFILL\_AUTH} = REFILL_AUTH \left|\left| M^{'} \right|\right| H\left(M^{'}\right)$$

De façon similaire à la procédure d'authentification, la passerelle enregistre l'identifiant de l'objet en l'associant à une procédure de rechargement sécurisée en cours et prévoit un délai maximal d'attente pour recevoir la réponse de l'objet.

A la réception du message, l'objet effectue les mêmes tests préliminaires que pour la procédure d'authentification. Le circuit PUF de l'objet est ensuite utilisé pour déterminer $P(C_n)$ et cette valeur est à son tour utilisée pour calculer $C(P(C_n); K = P(C_n))$ et comparer le résultat à la valeur envoyée par la passerelle. Si les deux valeurs concordent, la passerelle est authentifiée du point de vue de l'objet qui sauvegarde la réponse $P(C_n)$ en mémoire volatile. L'objet prépare ensuite son propre message d'authentification $N^{'}$ et l'envoie à la passerelle via un message REFILL_AUTH :

$$N^{'} = ID_{objet} \,||\, C(C_n;\ K\ =\ P(C_n))$$

$$DM_{REFILL\_AUTH}\ =\ REFILL\_AUTH \,||\, N' \,||\, H(N')$$

A la réception de ce message, la passerelle effectue les tests nécessaires pour s'assurer qu'il est valide et que le message d'authentification est correct. Si c'est le cas, l'objet est authentifié du point de vue de la passerelle. L'objet et la passerelle sont donc mutuellement authentifiés et partagent le secret $P(C_n)$ connu d'eux seuls. Ce secret est utilisé comme clé pour un algorithme de chiffrement symétrique, l'algorithme AES par exemple, qui est utilisé pour protéger le reste de la procédure de rechargement.

Après l'authentification mutuelle et l'établissement d'un canal de communication chiffré entre l'objet et la passerelle, la passerelle peut utiliser les messages INIT et CHALL de l'étape d'enregistrement. L'objet traitera ces messages comme lors de l'enregistrement et répondra avec des messages RESP. Tous ces échanges sont effectués de manière chiffrée pour garantir la confidentialité des données d'authentification ainsi générées. La passerelle dispose d'une contrainte supplémentaire dans le choix du challenge initial : il doit être supérieur à la valeur courante du compteur de l'objet. La passerelle ne peut pas faire reculer le compteur pour préserver la protection anti-rejeu du protocole. Une fois que la passerelle a généré un nombre suffisant de nouveaux CRP, elle met fin à l'étape de rechargement sécurisé en envoyant un message END à l'objet. Après ce message, la passerelle et l'objet doivent supprimer le secret partagé de leur mémoire.

Pour synthétiser, nous avons conçu un nouveau protocole d'authentification basé sur la technologie PUF qui permet de générer des données d'authentification liées au matériel. Ce protocole permet d'effectuer des authentifications mutuelles entre un objet connecté contraint et une passerelle de supervision. Il permet également de générer de nouvelles données d'authentification au cours de la vie de l'objet grâce à une étape de rechargement sécurisé, pour éviter un épuisement prématuré des données d'authentification. Cependant, ce protocole ne gère pas la confidentialité des données et la persistance de l'authentification par exemple. Dans l'optique de proposer une solution de sécurisation complète pour les objets connectés contraints, prenant en charge l'authentification, la confidentialité et l'intégrité ; nous avons travaillé à l'intégration de notre protocole dans TLS.

4.4.3 *Protocole PUF-TLS : Extension du protocole TLS avec une PUF*

Avant de présenter notre démarche d'intégration, nous allons présenter le fonctionnement du protocole TLS [Res18], un protocole éprouvé et largement utilisé pour l'établissement de communications sécurisées entre un client et un serveur. Dans le vocabulaire TLS, le client est l'équipement qui initie la communication vers le serveur, c'est-à-dire l'équipement avec lequel il souhaite échanger. Au sein du protocole TLS sont définis trois sous-protocoles qui sont utilisés pour établir une communication sécurisée :

— le *Handshake Protocol*, qui défini les options de communication et les secrets partagés selon les fonctionnalités supportées par le client et le serveur,
— le protocole *Change Cipher Spec*, qui gère l'installation des secrets lors du *Handshake Protocol*, et
— le *Record Protocol*, qui est en charge de l'échange des données applicatives entre le client et le serveur en assurant leur confidentialité et leur intégrité grâce aux secrets précédemment installés.

C'est dans le *Handshake Protocol* qu'ont lieu les procédures d'authentification, c'est donc sur ce protocole que nous nous sommes concentrés et que nous avons choisi d'intégrer notre protocole d'authentification basée sur la technologie PUF.

Le *Handshake Protocol* est lui-même divisé en quatre parties, qui sont illustrées par la figure 4.11.

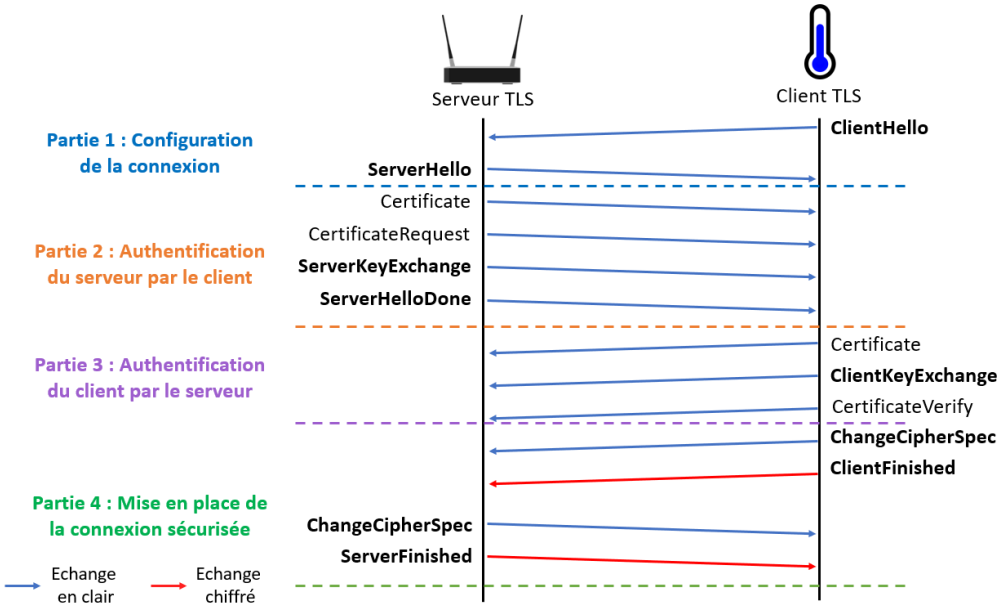

FIGURE 4.11 – Les quatre parties du Handshake Protocol de TLS.

Dans la première partie, le client et le serveur doivent s'accorder sur les paramètres de la communication. Pour ce faire, le client partage au serveur la liste des paramètres qu'il prend en charge via le message *Client Hello*. Ce message est principalement constitué de la liste des suites de chiffrement (*ciphersuite* en anglais) supportées par le client. Des informations supplémentaires peuvent également être incluses grâce à l'utilisation de la fonctionnalité d'extension offerte par la spécification TLS [Don11]. A la réception de ce message, le serveur

compare les paramètres supportés par le client aux paramètres qu'il accepte et supporte. Si aucune configuration ne convient au serveur, il peut fermer la communication. Le plus souvent toutefois, le serveur sélectionne une configuration proposée par le client et fait connaître son choix via le message *Server Hello*. La configuration inclut, notamment, la suite de chiffrement qui va être utilisée pour authentifier puis sécuriser la communication.

La deuxième partie consiste en l'authentification du serveur et l'envoie de son secret. La manière dont l'authentification est effectuée dépend de la suite de chiffrement sélectionnée. Cela a un impact sur les messages utilisés lors de cette phase. Par exemple, avec les méthodes d'authentification RSA le serveur utilise le message *Certificate* pour transmettre un certificat lié à son identité, alors qu'avec les méthodes d'authentification PSK qui n'utilisent pas de certificat, ce message n'est pas utilisé. Symétriquement, la troisième partie du *Handshake Protocol* consiste en l'authentification du client et l'envoie de son secret au serveur.

La dernière partie est l'installation et la vérification du secret de session. A cette étape, le client et le serveur ont normalement les informations nécessaires pour calculer un secret de session à partir, notamment, des secrets intermédiaires échangés lors des parties précédentes. Après vérification de la validité du secret, le *Handshake Protocol* se conclut et la communication sécurisée démarre entre le client et le serveur.

Dans notre démarche d'intégration du protocole d'authentification PUF au protocole TLS, nous avons respecté le déroulement des étapes présentées précédemment. Nous avions aussi pour but de minimiser le nombre de messages et la quantité totale de données échangées pour proposer une solution faiblement consommatrice de ressources, adaptée aux objets connectés contraints. Nous avons utilisé pour l'authentification avec notre protocole PUF uniquement les messages *ServerKeyExchange* et *ClientKeyExchange*, en plus des messages obligatoires au bon déroulement du protocole. L'ensemble des messages utilisés est en gras sur la Figure 46. Nous avons également utilisé le mécanisme d'extension de TLS pour transmettre l'identifiant de l'objet lors l'ouverture de la communication. Une fois le protocole PUF intégré, nous avons déclaré de nouvelles suites de chiffrement utilisant ce protocole pour l'authentification et des algorithmes reconnus, AES et SHA-2, pour assurer la confidentialité et l'intégrité.

Nous avons donc intégré notre protocole d'authentification de la manière suivante dans le *Handshake Protocol* de TLS :

1. l'objet connecté ouvre une communication TLS avec la passerelle et précise son identifiant en utilisant l'extension TLS dédiée.
2. la passerelle vérifie qu'elle connaît l'identifiant de l'objet, sélectionne une suite de chiffrement utilisant le protocole PUF et envoie son message d'authentification dans le message *ServerKeyExchange*.
3. l'objet vérifie le message d'authentification de la passerelle et envoie le sien via le message *ClientKeyExchange*. Il choisit comme pré-secret la dernière réponse PUF utilisée lors de l'authentification.
4. la passerelle vérifie le message d'authentification de l'objet et elle choisit comme pré-secret la dernière réponse PUF utilisée.

Si tout s'est déroulé correctement, l'objet connecté et la passerelle sont mutuellement authentifiés à l'issue de la procédure. De plus, ils partagent un pré-secret commun qui leur permet de calculer un secret de session et de poursuivre la communication de manière sécurisée. De cette manière, il est possible de bénéficier à la fois des avantages de la technologie PUF en termes de faible coût d'implémentation et de stockage sécurisé de secrets et aussi des avantages du protocole TLS, qui est largement déployé et dont la capacité à sécuriser des communications est éprouvée. Nous pensons que cette utilisation de notre protocole PUF au sein du protocole TLS est une brique prometteuse dans la construction de solutions de communication à la fois sécurisées et abordables pour les objets connectés, quels que soit leurs contraintes.

4.5 SYNTHÈSE

Nous avons rappelé dans ce chapitre la problématique et l'objectif principal de cette thèse : permettre l'utilisation à la fois simple et sécurisée d'objets connectés hétérogènes pour les applications pervasives. Pour atteindre cet objectif, notre approche s'appuie sur une plateforme pervasive sécurisée permettant de gérer la sécurité hétérogène des objets connectés, de faciliter leur utilisation par les applications pervasives via une approche à services et d'évaluer le niveau de sécurité des technologies déployées sur les objets connectés pour aider les applications dans leurs choix de services. Cette approche est composée de deux éléments principaux : une extension pour plateforme pervasive dédiée à la gestion et l'évaluation de la sécurité des objets connectés et un protocole de communication sécurisé et léger, basé sur la technologie PUF.

En s'appuyant sur un modèle générique de plateforme pervasive utilisant l'approche à service pour faciliter l'utilisation des objets connectés, notre extension a pour objectif d'ajouter le support des fonctionnalités de sécurité hétérogènes des objets. Pour cela, nous utilisons un système de sous-modules de communications indépendants pour gérer les différents protocoles de communication utilisés par les objets connectés et leurs configurations de sécurité. L'extension comprend aussi un modèle de données conçu pour représenter précisément la sécurité protocolaire et matérielle des objets connectés. Basé sur cette description précise et concrète, l'extension expose avec chaque service une description abstraite et synthétique des caractéristiques de sécurité assurées par l'objet sous-jacent. Nous proposons également une démarche d'évaluation du niveau de sécurité basée sur la description de sécurité concrète. Cette approche a fait l'objet d'une publication [Des+22a] à la conférence internationale IEEE EDGE 2022.

Notre protocole de communication a pour objectif de proposer un compromis entre un niveau de sécurité correct et un coût de déploiement abordable, en particulier du côté des objets connectés. Pour cela, il utilise, notamment, la technologie PUF pour faire de l'authentification mutuelle entre un objet connecté équipé d'un circuit PUF et un équipement de supervision. Les étapes du protocole sont pleinement intégrées au cycle de vie d'un objet connecté, pour assurer l'authentification de sa fabrication à sa fin de vie. Enfin, le protocole permet une gestion flexible des données d'authentification, qui facilite le passage à l'échelle du nombre d'objets sécurisés, grâce à une procédure dédiée

permettant la génération sécurisée de nouvelles données d'authentification durant la vie de l'objet. La version de base de notre protocole assure uniquement l'authentification mutuelle d'un échange. Nous avons également conçu une version améliorée qui permet de faire un échange authentifié de secret partagé et qui est compatible avec le protocole TLS. Cela permet d'aboutir à une solution complète de communication sécurisée. Nous avons présenté le protocole d'authentification PUF dans une publication [Des+21] à la conférence internationale IE 2021 et dans un article [Des+22b] publié dans le *Journal of Ambient Intelligence and Smart Environments*.

Nous avons réalisé des démonstrateurs de ces contributions, pour valider leur fonctionnement et évaluer leurs performances. Ces travaux sont présentés dans le chapitre suivant.

5 RÉALISATION ET EXPÉRIMENTATIONS

Dans ce chapitre, nous présentons nos travaux sur l'implémentation puis l'évaluation de l'architecture de plateforme pervasive munie de son extension de sécurité et du protocole de communication sécurisé et léger présentés dans le chapitre précédent. Nous terminons par un exemple d'utilisation de notre plateforme dans un scénario pervasif avec plusieurs objets connectés hétérogènes, dont un objet utilisant le protocole que nous proposons. Cet exemple illustre la synergie de nos contributions pour faciliter l'utilisation sécurisée des objets connectés.

5.1 PLATEFORME PERVASIVE SÉCURISÉE

5.1.1 *Implémentation*

Pour valider l'architecture proposée et dans le but d'évaluer les performances de notre approche, nous avons développé un démonstrateur constitué :

— d'une plateforme pervasive avec son extension de sécurité, basée sur l'architecture présentée dans la section 4.3 ;
— de plusieurs objets connectés, de type capteur, effectuant de la remontée périodique de données via différents protocoles.

Nous avons développé la plateforme avec une approche microservices. Chaque module de notre architecture est implémenté par une application développée en Java avec Quarkus [1], un *framework* adapté au développement de services et microservices Web. Les modules exposent leurs fonctionnalités aux applications pervasives via des interfaces accessibles en utilisant le protocole sécurisé HTTPS. Pour mettre en place ce protocole sécurisé, nous avons créé une autorité de certification pour la plateforme, puis nous avons généré pour chaque interface une clé privée et un certificat signé par l'autorité de certification. Les applications peuvent ainsi authentifier les interfaces de la plateforme et échanger de manière sécurisée, simplement en faisant confiance au certificat de l'autorité associée à la plateforme.

La communication entre les modules est assurée par des files de messages, gérées par un service Kafka [2]. Le système de files de messages permet aux modules de partager des informations sans avoir besoin de connaître les autres modules présents, ce qui facilite le déploiement de nouveaux modules et l'évolutivité de la plateforme. La solution Kafka a été choisie en raison de ses performances et de sa capacité à passer à l'échelle, des qualités nécessaires pour un bon fonctionnement dans des environnements connectés et pervasifs. Kafka gère également la sécurisation des échanges en s'appuyant sur des solutions éprouvées, telles que le protocole sécurisé TLS. Concernant le module Registre, après avoir passé en revue des solutions existante de découverte et de mise à

1. `https://quarkus.io/`.
2. `https://kafka.apache.org/`.

disposition de services, telles que Consul [3] et Apache Zookeeper [4], nous avons choisi d'implémenter notre propre solution pour disposer d'une plus grande flexibilité. Cela nous a également permis de facilement déployer notre modèle de données permettant, notamment, la description des caractéristiques de sécurité d'un objet, telle que décrit dans la section 4.3.3. Toutefois, l'intégration de notre modèle de description de sécurité dans les solutions mentionnées précédemment est souhaitable et pourra faire l'objet de travaux futurs.

Nous avons également travaillé sur la mise au point d'objets connectés qui se connectent à la plateforme pour remonter des données. Nous avons chercher à construire un panel d'objets couvrant des situations variées : une communication non sécurisée, une communication avec un niveau d'authentification simple, des cibles plus ou moins contraintes et plusieurs protocoles de communication. Ces objets ont permis de tester et valider les fonctionnalités de la plateforme au fur et à mesure du développement. Pour cela, nous avons eu recours d'un part à des scripts et d'autre part à des cartes de développement embarquées pour objet connecté.

Nous avons utilisé des scripts pour simuler les comportements attendus d'un objet : son enregistrement auprès de la plateforme, puis la remontée périodique de données. Cela nous a permis de tester les fonctionnalités de notre plateforme et de valider leur bon fonctionnement. Un autre avantage de l'utilisation d'objets simulés est la possibilité d'invoquer facilement de nombreuses instances pour tester la plateforme en charge.

Toutefois, nous avons également montré qu'il est possible, pour notre plateforme, de prendre en charge des objets concrets. Pour cela, nous avons développé des applications embarquées sur la carte de développement IoT *MT3620 Starter Kit v1*, visible sur la figure 5.1. Cette carte dispose d'une connectivité Wi-Fi et d'un capteur gyroscopique. Elle fait partie du programme Microsoft Azure Sphere [5], qui vise à développer des solutions IoT sécurisées. Nous avons développé sur cette cible un total de trois applications. Chaque application s'enregistre auprès de la plateforme, puis effectue une remontée périodique de données à l'aide d'un protocole de communication particulier. Nous avons utilisé les protocoles de communication HTTP, HTTPS et MQTT pour tester le support de ces différents protocoles et de leurs caractéristiques de sécurité par la plateforme.

3. `https://www.consul.io/`.
4. `https://zookeeper.apache.org/`.
5. `https://azure.microsoft.com/fr-fr/products/azure-sphere/`.

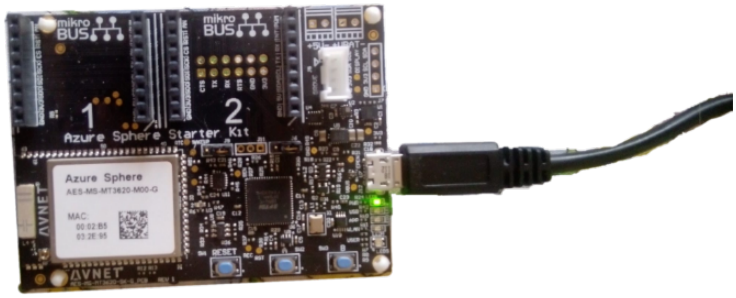

FIGURE 5.1 – Le kit de développement 'MT3620 Starter Kit v1'.

Au final ce démonstrateur, nous a permis d'évaluer les apports de notre approche pour faciliter l'utilisation sécurisée des objets connectés par les applications pervasives.

5.1.2 *Evaluation*

Pour évaluer notre plateforme pervasive et ses fonctionnalités, nous nous sommes placés dans le scénario d'exemple présenté dans la section 2.1, sur la gestion de la température intérieure d'une habitation. Nous avons ensuite estimé les efforts nécessaires au développement et à la maintenance de l'application pervasive de gestion de la température, d'abord sans puis avec notre plateforme. Nous nous sommes, en particulier, concentré sur les étapes suivantes :

— le développement initial de l'application pervasive,
— le support de nouveaux objets connectés et/ou de nouveaux protocoles de communication, une fois l'application développée et déployée,
— la mise à jour d'un protocole.

Développement initial de l'application pervasive

Commençons par le développement initial d'une application pervasive. Nous avons vu, dans la section 2.3, que c'est une tâche complexe du fait des nombreuses exigences à respecter pour implémenter la vision pervasive. Il faut, notamment, être capable de gérer la récupération de données depuis des sources diverses et hétérogènes : différents capteurs de température répartis dans une habitation, pouvant provenir des fournisseurs différents. Cela demande de développer du code dédié à cette récupération de données, du code en charge de se connecter aux différents objets connectés disponibles et gérer la collecte des données selon les spécificités de chaque objet : ses interfaces et son format de données. La communication avec ces objets nécessite le support des protocoles de communication adéquats. Pour chaque protocole, il est probable qu'il soit nécessaire d'intégrer à l'application un à plusieurs composants logiciels dédiés, qui doivent ensuite être utilisés correctement, ce qui peut demander une certaine expertise. Cette intégration n'est pas neutre car elle influence sur la taille finale du code de l'application, la rendant plus importante, et elle peut créer des dépendances à certains composants logiciels qui complexifient le déploiement de l'application. De plus, la gestion de la configuration sécurisée de chaque protocole est également à la charge de l'application, pour garantir

des échanges authentifiés, confidentiels et intègres avec les objets connectés en fonction de leurs capacités. Enfin, c'est à l'application d'évaluer en autonomie la confiance qu'elle accorde à chaque objet. Ces tâches demandent une expertise en sécurité. Il faut allouer du temps de développement et des ressources à l'élaboration de ce code support. En conséquence, il est possible que cela ralentisse le développement du code fonctionnel de l'application, la partie qui apporte le service et la valeur ajoutée pour les utilisateurs.

En utilisant notre plateforme pervasive maintenant, c'est cette dernière qui est en charge de communiquer avec les objets connectés et de récupérer leurs données. La plateforme gère le support des différents protocoles de communication utilisés et leur utilisation sécurisée. Pour récupérer les données d'un objet, l'application pervasive a uniquement besoin de communiquer avec la plateforme via un protocole de communication unique et fixe dans le temps : le protocole sécurisé HTTPS. De surcroît, l'extension de sécurité de notre plateforme associe à chaque objet un niveau de sécurité basé sur les technologies de sécurisation utilisées par cet objet, dans un format facilement utilisable par les applications. Cela réduit considérablement les efforts de développement engagés dans du code support pour communiquer et choisir des objets sécurisés.

Support de nouveaux objets connectés et/ou de protocoles de communication

Concernant l'ajout de nouveaux objets une fois le développement initial de l'application terminé, sans plateforme pervasive, il est nécessaire de développer du nouveau code support pour gérer la communication avec ces nouveaux objets et leur sécurité, puis de modifier la partie fonctionnelle du code applicatif pour utiliser ces nouveaux objets. Ces nouveaux développements peuvent être plus ou moins longs et coûteux selon la quantité de nouveaux objets à supporter et leur complexité. De plus, si certains de ces nouveaux objets utilisent un ou plusieurs nouveaux protocoles de communication, les efforts de développement seront plus importants encore.

Avec notre plateforme pervasive, le support de nouveaux objets et de nouveaux protocoles de communication est une opération presque transparente pour l'application pervasive. C'est le code de la plateforme qui est mis à jour pour gérer les nouveaux objets et les nouveaux protocoles si nécessaires, avec leurs configurations de sécurité. La sécurité des nouveaux objets est automatiquement évaluée grâce au modèle de l'extension de sécurité. L'application pervasive voit simplement apparaître de nouveaux services, attachés aux nouveaux objets, avec des informations sur leur niveau de sécurité. Il peut être nécessaire de modifier légèrement le code fonctionnel de l'application pour faire usage de ces nouveaux services.

Mise à jour d'un protocole de communication

Intéressons-nous pour finir au cas de la mise à jour d'un protocole pour des raisons de sécurité. Cela peut faire suite à la découverte d'une vulnérabilité dans le protocole en question ou bien à la fin du support d'une version du protocole, par exemple. Il est alors nécessaire d'effectuer une mise à jour des composants logiciels qui gèrent ce protocole. Selon l'importance de la mise à jour, les efforts peuvent être plus ou moins conséquents. Par exemple, s'il s'agit d'une mise à jour majeure qui n'est pas la rétro-compatible avec les versions antérieures, il

est nécessaire de modifier le code support associé à la gestion de ce protocole dans un premier temps, puis le code fonctionnel faisant appel à ce code support. C'est une opération qui peut se révéler très complexe selon la taille du code et l'ampleur des modifications à effectuer. En attendant le développement et le déploiement de la mise à jour, il est possible que l'application pervasive continue d'utiliser le protocole vulnérable, même si des objets plus sécurisés sont disponibles.

Avec notre plateforme, seule la plateforme doit être mise à jour ; car c'est elle qui centralise la gestion des protocoles de communication. Une fois la nouvelle version du protocole déployée sur la plateforme, cette dernière est utilisée lorsque nécessaire pour communiquer avec les objets. Les services utilisant ce protocole sont mis à jour et l'application en bénéficie de manière transparente en utilisant l'interface de la plateforme appropriée. Aucune modification du code de l'application n'est nécessaire. De plus, durant le développement de la mise à jour pour la plateforme, il est possible de dégrader le niveau de sécurité du protocole vulnérable via une mise à jour, beaucoup plus simple, de l'algorithme d'évaluation de la sécurité de notre plateforme. Cela aura pour conséquence de faire baisser le niveau de sécurité des services offerts par les objets utilisant ce protocole. Si l'application utilise cette information pour le choix des services qu'elle utilise, elle se reconfigurera automatiquement pour utiliser des services plus sécurisés, sans nécessiter de modification de son code.

Pour synthétiser, nous pensons que notre plateforme pervasive et ses fonctionnalités facilite le développement d'applications pervasives sécurisées par rapport à des démarches de développement classiques en :

— facilitant l'accès simple et sécurisé aux objets connectés lors du développement initial d'une application pervasive ;
— minimisant les efforts nécessaires à la prise en compte et l'utilisation de nouveaux objets connectés ;
— rendant transparent pour les applications l'ajout du support de nouveaux protocoles, ou la mise à jour des protocoles existants.

5.1.3 *Comparaison avec des travaux similaires*

Comme nous l'avons déjà souligné, la recherche est active sur le sujet des plateformes pour supporter le développement des applications IoT et pervasives. Nous comparons dans cette section notre plateforme muni de son extension de sécurité avec d'autres plateformes pervasives récentes utilisant un approche à services, présentées dans la section 3.2, pour identifier leurs points communs ainsi que leurs différences dans la gestion des objets connectés et de leur sécurité hétérogène.

La plateforme Hydra présentée par M. Eisenhauer *et al.* [ERA10] base la description des objets gérés sur des ontologies, qui se veulent plus génériques que le modèle que nous avons développé mais possiblement moins précises, en particulier sur la description des technologies de sécurité, et plus difficiles à faire évoluer. La plateforme dispose d'une couche dédiée à la sécurité qui a pour but de garantir la confidentialité et l'intégrité des données, ainsi que l'authentification. Toutefois, nous n'avons pas pu déterminer si cette couche sécurise les communications entre les applications et la plateforme, les commu-

nications entre la plateforme et les objets connectés ou bien les deux. Dans le cas de notre plateforme, les deux types de communication sont sécurisés, avec une flexiblité sur la partie objets connectés pour être en mesure de s'adapter à des configurations de sécurité hétérogènes. Enfin, les services sont décrit aux applications en utilisant comme base les Web Services, étendues par des ontologies. Nous n'avons pas été en mesure de déterminer précisemment le contenu de ces extensions et leur potentiel lien avec une description de la sécurité des objets connectés fournissant les services, comme nous avons proposé dans notre approche.

La plateforme ubiREST présentée par M. Caporuscio *et al.* [Cap+14] est basée, similairement à notre plateforme, sur une architecture REST augmentée d'un registre de services pour permettre aux applications pervasives de facilement rechercher et utiliser des services, services qui sont fournis par des objets connectés. Les services sont décris avec une approche principalement fonctionnelle, nous n'avons pas trouvé d'informations sur la prise en compte de propriétés non-fonctionnelles relatives à la sécurité comme nous proposons dans notre plateforme. Du côté des objets connectés, le but de la plateforme est de supporter un nombre aussi varié que possible de protocoles de communication, tout en masquant cette hétérogénéité aux applications pervasives. Pour cela, les auteurs ont adopté une approche modulaire similaire à ce que nous proposons. Toutefois, la sécurité hétérogène associée aux divers protocoles supportés ne semble pas prise en compte, contrairement à ce qui est fait dans notre plateforme.

Pour finir, la plateforme In.IoT présentée par M. da Cruz *et al.* [da +21] utilise, comme notre plateforme, une architecture à base de microservices pour implémenter l'approche à services qui permet d'abstraire les fonctionnalités des objets connectés. Chaque protocole de communication supporté par la plateforme est géré par un module dédié. Un registre de services est également présent dans l'architecture, implémenté sous forme d'un module. La sécurité est un des enjeux clé de cette plateforme avec, notamment, l'authentification des objets et des utilisateurs de la plateforme, ainsi que la gestion des actions qu'ils peuvent effectuer ou non sur la plateforme. Toutefois, l'authentification des objets semble reposer principalement sur la connaissance d'un secret partagé qui est distribué par la plateforme lors de l'enregistrement de l'objet via le protocole de communication utilisé. Cela suppose que ce premier échange est sécurisé et que l'objet est capable de stocker le secret d'authentification dans un espace mémoire sécurisé. Ces informations ne semblent pas prises en compte par la plateforme. De plus, seule la description fonctionnelle des services semble accessible via la plateforme. Il n'est pas fait mention de la présence de propriétés non-fonctionnelles concernant la qualité de la sécurité associée à un service donné, comme nous le proposons dans notre plateforme.

Pour synthétiser, notre proposition de plateforme sécurisée utilise une approche assez répandue basée sur l'architecture orientée services [Pap03] avec l'utilisation de services pour abstraire les objets connectés et faciliter leur usage, ainsi qu'un registre pour faciliter la recherche de ces services. Nous avons cependant étendu cette approche pour mieux prendre en compte les caractéristiques de sécurité hétérogènes des objets connectés et, en particulier, nous avons étendu le modèle de description d'un service pour afficher ces caractéristiques

de sécurité aux applications pervasives, leur permettant ainsi d'utiliser ces nouveaux paramètres pour choisir les services utilisés.

5.2 PROTOCOLE DE COMMUNICATION SÉCURISÉ ET LÉGER

5.2.1 *Implémentation du protocole d'authentification mutuelle*

Pour valider notre protocole d'authentification mutuelle basé sur la technologie PUF, nous avons développé un démonstrateur composé :

— d'un objet connecté contraint, équipé d'un circuit PUF,
— d'une passerelle,
— et d'un registre d'authentification.

Avec ces éléments, il est possible de tester l'enregistrement de l'objet, son authentification et la génération sécurisée de nouvelles données d'authentification.

Nous avons utilisé deux équipements pour mettre en place notre démonstrateur : une carte de développement LPC55S69 [6] de NXP dans le rôle de l'objet connecté et un ordinateur de bureau qui prend alternativement les rôles de registre d'authentification, puis de passerelle. La communication entre ces équipements est assurée par un lien série qui possède les caractéristiques suivantes : une vitesse de transfert de 115 200 bauds, 8 bits de données et pas de bit de parité. L'architecture détaillée du démonstrateur est illustrée par la figure 5.2.

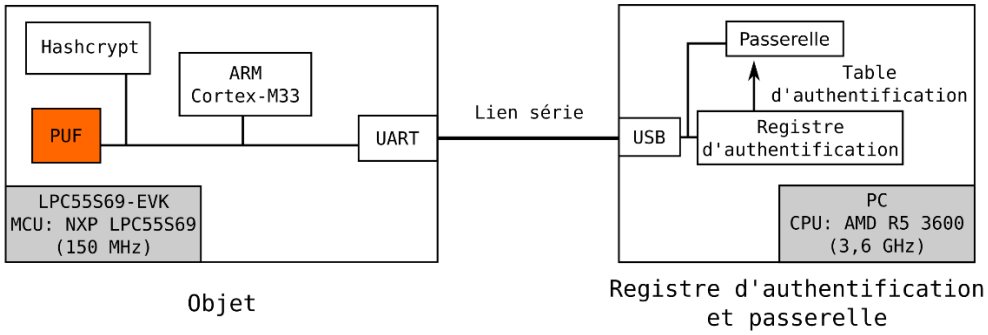

FIGURE 5.2 – Architecture du démonstrateur pour le protocole PUF.

Le microcontrôleur LPC55S69 de NXP est basé sur un processeur ARM Cortex M33 avec une cadence d'horloge modeste pour une consommation énergétique faible, adaptée à des cas d'usage IoT. Il possède également un sous-système PUF basé sur la technologie SRAM PUF, avec des mécanismes de correction d'erreurs intégrés. La SRAM PUF étant une PUF faible, nous avons utilisé dans notre démonstrateur le travail de Bhargava *et al.* [BM14] pour concevoir un équivalent d'une PUF forte en combinant la SRAM PUF avec un algorithme de chiffrement symétrique. Nous avons utilisé l'algorithme de chiffrement AES en mode *Electronic Code Book* (ECB) avec une taille d'entrée/sortie de 128 bits et une taille de clé de 128 bits. Le LPC55S69 dispose d'un accélérateur cryptographique, nommé Hashcrypt, qui implémente cette configuration. Cependant ce choix

6. `https://www.nxp.com/design/development-boards/lpcxpresso-boards/lpcxpresso55s69-development-board:LPC55S69-EVK`.

n'est pas obligatoire, un autre algorithme et d'autres configurations peuvent être utilisés en prenant soin d'examiner les effets en termes de sécurité et de performances. Il résulte de cette combinaison un sous-système émulant le comportement d'une PUF forte acceptant des challenges de 128 bits et produisant des réponses de 128 bits. L'architecture de ce sous-système est présentée par la figure 5.3.

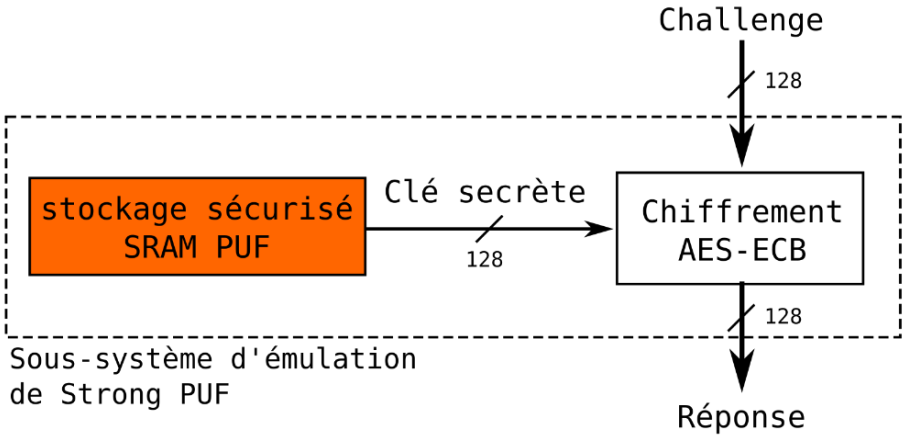

FIGURE 5.3 – Emulation d'une PUF forte avec une PUF faible.

Dans cette architecture, la SRAM PUF fait office de stockage sécurisé pour la clé secrète utilisée par l'algorithme AES. Les challenges sont envoyés au sous-système AES, qui les chiffre pour produire les réponses. En utilisant ce sous-système, nous avons implémenté le protocole d'authentification tel que nous l'avons décrit dans la section 4.4.2. Le logiciel embarqué de l'objet a été écrit en langage C, en utilisant les interfaces fournies par NXP via son kit de développement.

Le logiciel sur PC qui implémente le registre d'authentification et la passerelle, a été écrit en langage Python 3. Nous avons utilisé le module PySerial pour communiquer avec l'objet via la liaison série. Dans notre application, la partie registre d'authentification commence par enregistrer l'objet et construire la table d'authentification, puis la partie passerelle effectue les authentifications. La table d'authentification est partagée localement. Dans un cas d'usage concret, il faut veiller à l'échange sécurisé de ces données et à leur stockage pour limiter les risques de vol de données.

Pour synthétiser, nous avons réalisé un démonstrateur permettant de valider l'utilisation de notre protocole d'authentification mutuelle utilisant la technologie PUF entre un objet connecté contraint, représenté par une carte de développement embarquée et un équipement plus puissant. Une fois cette première étape de fonctionnement validée, nous avons travaillé à la réalisation d'un autre démonstrateur utilisant le protocole PUF-TLS, pour montrer que nous pouvions intégrer notre protocole d'authentification mutuelle basé sur la technologie PUF au sein d'une librairie TLS. Nous détaillons cette réalisation dans la section suivante.

5.2.2 *Implémentation du protocole de communication sécurisé PUF-TLS*

Pour valider l'intégration de notre protocole d'authentification PUF dans le protocole sécurisé TLS, nous avons réalisé un second démonstrateur composé :

— d'une application côté objet connecté, remontant des données en utilisant le protocole TLS ;
— d'une application côté passerelle, acceptant des connexions entrantes via le protocole TLS.

Ces deux applications utilisent une version modifiée de la librairie GnuTLS[7] dans laquelle nous avons implémenté, notamment, notre algorithme d'authentification utilisant la technologie PUF. L'architecture globale du démonstrateur est illustrée par la figure 5.4.

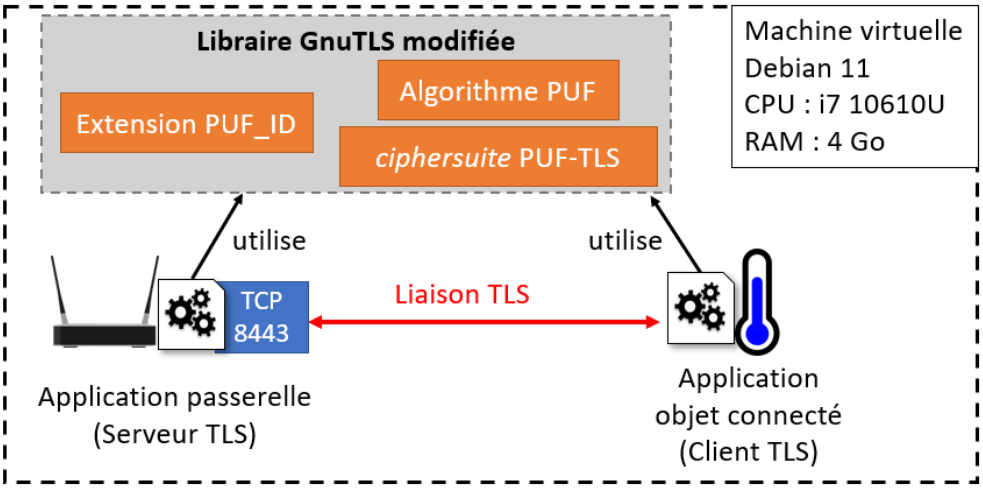

FIGURE 5.4 – Architecture du démonstrateur pour le protocole PUF-TLS.

Nous avons choisi la librairie GnuTLS comme support pour l'intégration de notre protocole d'authentification et le développement de notre démonstrateur ; car c'est un projet libre et en source ouverte qui implémente le protocole TLS et les fonctions support associées.

En plus du protocole d'authentification, nous avons ajouté dans la librairie TLS une nouvelle extension nommée PUF_ID et une nouvelle suite de chiffrement nommée PUF-TLS. La nouvelle extension TLS est utilisée conjointement au protocole d'authentification et permet au client TLS, c'est-à-dire un objet connecté, de transmettre son identifiant dans sa première requête, tel que décrit dans la section 4.4.3. La nouvelle suite de chiffrement, ou *ciphersuite*, permet d'utiliser notre algorithme basé sur la technologie PUF pour l'échange de clés authentifié en début de connexion. Pour les algorithmes de chiffrement et de contrôle d'intégrité, nous avons choisi des algorithmes standards : respectivement l'algorithme AES et l'algorithme SHA-256. Ces algorithmes sont largement utilisés, ils sont implémentés nativement par la librairie GnuTLS et leur usage de ressources est compatible avec des cibles embarquées contraintes.

Avec ces éléments, nous sommes capables d'établir une connexion sécurisée entre l'application côté objet connecté et l'application côté passerelle, en utilisant une authentification mutuelle basée sur la technologie PUF.

7. `https://www.gnutls.org/`.

5.2.3 *Evaluation*

Pour évaluer le protocole PUF, nous avons tout d'abord validé ses caractéristiques de sécurité par une analyse théorique prenant en compte plusieurs types d'attaques. Ensuite, nous avons déterminé de manière concrète les performances de notre protocole dans sa version de base et dans sa version TLS, en utilisant les démonstrateurs présentés dans les sections précédentes. Nous avons évalué l'empreinte mémoire, le coût de communication et le temps d'exécution de différentes étapes du protocole. Les résultats et les analyses de ces évaluations sont présentés dans cette section.

5.2.3.1 *Sécurité du protocole*

Avant de présenter l'analyse de sécurité du protocole, nous définissons le périmètre que nous avons étudié ainsi que le modèle d'attaque choisi. Nous avons mené notre analyse sur les phases d'authentification et de rechargement sécurisé du protocole. Notre situation d'analyse est la communication entre un objet connecté contraint équipé d'un circuit PUF et un équipement de supervision, une passerelle par exemple. La communication entre ces deux équipements est effectuée sur un canal ouvert et supposé non sécurisé. Nous travaillons sous l'hypothèse que la phase d'enregistrement de l'objet a été menée dans un environnement sécurisé, par une entité digne de confiance. Cette même entité gère le registre d'authentification dans lequel est stocké un ensemble initial de données d'authentification liées à la PUF de l'objet. Enfin, nous supposons que la passerelle et le registre d'authentification sont capables de communiquer entre eux de manière sécurisée, via un protocole sécurisé classique comme le protocole TLS.

Concernant le profil d'attaquant étudié, nous avons choisi un attaquant suivant le modèle Dolev-Yao [DY81], qui a donc une grande liberté d'action sur le canal de communication : il peut écouter le canal et lire les messages échangés, choisir d'intercepter certains messages, d'en rejouer voire d'en injecter de nouveaux à sa guise. Le but principal de l'attaquant que nous avons étudié est le contournement de l'authentification ; c'est-à-dire la capacité à compléter avec succès une procédure d'authentification en usurpant l'identité de l'objet ou de la passerelle. Nous avons également examiné quelques objectifs secondaires, comme la perturbation du bon fonctionnement des opérations du protocole.

Dans la situation décrite et avec les hypothèses présentées, nous avons analysé le protocole. Il ressort de notre analyse les caractéristiques de sécurité suivantes :

— **Authentification de l'objet connecté légitime uniquement**. Un équipement est authentifié par la passerelle uniquement s'il est capable de fournir un message d'authentification valide. Ce message est composé à partir de données issues du circuit PUF de l'objet, à savoir les réponses du circuit PUF. Aucune réponse n'est échangée en clair sur le canal de communication. Sans accès légitime au circuit PUF, un attaquant ne peut pas exploiter d'information particulière lui permettant de deviner la valeur des réponses. Il doit s'en remettre au tirage aléatoire sur l'ensemble des valeurs de réponses possibles. La probabilité de succès est très faible si les réponses sont de taille suffisante, supérieures ou égales à 128 bits

par exemple. Nous considérons ainsi que seul un objet légitime peut s'authentifier avec succès.

— **Authentification d'une passerelle légitime uniquement**. Une passerelle est authentifiée par l'objet connecté uniquement si elle est capable de fournir un message d'authentification valide. Ce message est composé à partir de données issues du circuit PUF de l'objet, à savoir les réponses du circuit PUF. A l'exception de l'objet connecté qui possède le circuit PUF, ces réponses sont seulement connues du registre d'authentification puis d'une passerelle approuvée par le registre. Sans accès aux données du registre ou d'une passerelle approuvée, un attaquant doit s'en remettre au tirage aléatoire sur l'ensemble des valeurs de réponses possibles. Comme précédemment, la probabilité de succès est très faible si les réponses utilisées sont de taille suffisante. Nous considérons donc que seule une passerelle approuvée par le registre d'authentification, donc considérée comme légitime, peut s'authentifier avec succès.

— **Résistance aux attaques par rejeu**. Une attaque par rejeu consiste, pour l'attaquant, à enregistrer les messages d'une session d'authentification, puis à rejouer un ou plusieurs de ces messages pour valider une nouvelle authentification. Cela n'est pas possible dans notre protocole. Du côté de l'objet connecté, un compteur anti-rejeu est mis à jour à chaque authentification réussie. Un message rejoué ne passera pas le contrôle du compteur anti-rejeu et sera donc rejeté. Du côté de la passerelle, un mécanisme similaire est en place, avec l'évolution de l'état des entrées de la table d'authentification (*success*, *fail*) au fur et à mesure des utilisations. Ainsi, un message valide rejoué ne pourra pas être traité une seconde fois.

— **Protection contre les attaques de modélisation du circuit PUF.** Les attaques de modélisation sont un type d'attaque contre les PUF qui consiste à générer un modèle numérique d'un circuit PUF donné. Ce modèle peut ensuite prédire, avec une probabilité élevée, la réponse du circuit à une entrée donnée [RS14]. La génération du modèle s'appuie sur des techniques d'apprentissage machine, qui nécessitent de collecter des réponses du circuit PUF et les entrées associées, c'est-à-dire des CRP. Pour prévenir ce type d'attaque, les réponses ne sont jamais échangées en clair dans notre protocole. Nous pensons que cela peut rendre plus délicat la modélisation du circuit PUF utilisé par notre protocole.

— **Protection contre les attaques par déni de service.** L'attaque par déni de service ne vise pas le contournement du mécanisme d'authentification, mais plutôt la perturbation du fonctionnement du protocole pour gêner son usage par des acteurs légitimes. Un attaquant peut, par exemple, cibler un équipement avec un nombre important de messages pour monopoliser ses ressources ou provoquer une surconsommation d'énergie, qui peut modifier la durée de vie d'un objet connecté fonctionnant sur batterie. Pour limiter ce type d'attaque du côté des objets connectés, nous avons choisi de réaliser l'authentification de la passerelle en premier. De cette manière, les objets connectés peuvent très rapidement filtrer les messages non authentiques et limiter les traitements associés. Du côté de la passerelle, il est possible pour un attaquant de se faire passer pour un objet et de multiplier les demandes d'authentification, car ces demandes ne sont

pas authentifiées. Cela a pour effet de consommer des données d'authentification, ce qui peut conduire à un épuisement prématuré. Pour contrer ce type d'attaque, nous recommandons de mettre en place une surveillance des tables d'authentification, pour détecter et réagir à des rythmes de consommation anormaux, par la mise en quarantaine temporaire des objets concernés, par exemple.

5.2.3.2 *Empreinte mémoire*

Nous avons relevé l'espace mémoire occupé par les implémentations de notre protocole d'authentification PUF et par notre version modifiée de la librairie GnuTLS pour vérifier s'ils sont déployables ou non sur des objets connectés contraints, tels que nous les avons défini dans la section 2.2. Nous avons également cherché cette information pour d'autres implémentations de protocoles d'authentification basés sur la technologie PUF et nous avons rassemblés ces résultats dans le tableau 5.1.

Protocole	**Taille mémoire (Kio)**
Notre protocole d'authentification PUF	13
Protocole de A. Aysu *et al.* [Ays+15]	8,1
Protocole de F. Farha *et al.* [Far+21]	7
Protocole de K. Lounis et M. Zulkernine [LZ21]	20
Mbed TLS 1.3 avec authentification symétrique [RTB20]	23
Mbed TLS 1.3 avec authentification asymétrique [RTB20]	53
GnuTLS, version légère avec protocole PUF-TLS	9 304
GnuTLS, version légère sans protocole PUF-TLS	9 196

TABLE 5.1 – Empreinte mémoire de différents protocoles et librairies TLS.

En considérant un objet avec des contraintes mémoires de classe 1, c'est-à-dire environ 100 Kio d'espace mémoire dédié aux applications, le déploiement d'un des protocoles PUF présentés occuperait entre 10% et 20% de la mémoire. En comparant ces résultats avec la taille de la librairie Mbed TLS, la version avec uniquement le support de l'authentification symétrique a une taille environ deux fois plus importante que la plupart des protocoles PUF à l'exception de celui de Louis et Zulkernine. La version supportant l'authentification asymétrique est environ quatre fois plus volumineuse que notre protocole PUF et occuperait plus de 50% de la mémoire disponible sur un objet contraint. Ces résultats illustrent la nature légère des protocoles d'authentification basés sur la technologie PUF en comparaison à des solutions classiques pouvant occuper plusieurs centaines de Kio, voire plusieurs Mio.

Concernant d'ailleurs l'implémentation du protocole PUF-TLS dans la librairie GnuTLS, nous constatons que son empreinte mémoire est trop volumineuse par rapport aux contraintes mémoire que nous ciblons. Nous supposons que cela est principalement dû au fait que la librairie GnuTLS n'a pas été conçue

pour un déploiement sur cible embarquée, comme l'atteste une empreinte de 9,2 Mio pour sa version la plus légère. Nous avons obtenu cette version en désactivant le plus d'options possible lors de la compilation. Le choix d'une librairie TLS destinée aux cibles embarquées, comme la librairie Mbed TLS [8] ou la librairie WolfSSL [9], comme base pour implémenter notre protocole d'authentification PUF devrait résoudre ce problème d'empreinte mémoire. Nous n'avons toutefois pas eu le temps de réaliser ce portage durant la thèse pour tester cette hypothèse ; car l'intégration de nouveaux éléments dans ces librairies est plus difficile que dans la librairie GnuTLS.

5.2.3.3 *Coûts de communication*

Nous avons évalué le coût de communication de notre protocole d'authentification, dans sa version de base et dans sa version intégrée au protocole TLS. Par coût de communication, nous entendons la taille des messages, en octets, à échanger pour compléter un protocole. Selon le média de communication utilisé, l'émission de message peut être une opération plus coûteuse en énergie que la réception [MPL21]. Ainsi il est intéressant de chercher à minimiser ce coût, en particulier du côté de l'objet connecté qui peut être soumis à des contraintes énergétiques fortes comme le fonctionnement sur une batterie non-remplaçable et non-rechargeable.

Pour le protocole PUF, nous avons comparé ces résultats avec des protocoles similaires basés sur la technologie PUF. Nous avons également intégré les coûts de communication d'une librairie TLS pour permettre une comparaison avec les protocoles actuels. Lorsque cela était possible, nous avons indiqué séparément la taille des messages de l'objet et la taille des messages de l'authentificateur. L'authentificateur est le nom générique donné à l'équipement en communication avec l'objet et cherchant à l'authentifier, la passerelle dans notre modèle. Les données sur les coûts de communications sont présentées dans le tableau 5.2.

8. `https://www.trustedfirmware.org/projects/mbed-tls/`.
9. `https://www.wolfssl.com/`.

Protocole	**Coût de communication (octet)**	
Notre protocole PUF	Authentificateur	53
	Objet connecté	49
	Total	**102**
Protocole de A. Aysu *et al.* [Ays+15]	Authentificateur	32
	Objet connecté	271
	Total	**303**
Protocole de M. Barbareschi *et al.* [BDBM18]	Authentificateur	16
	Objet connecté	16
	Total	**32**
Protocole de F. Farha *et al.* [Far+21]	Authentificateur	108
	Objet connecté	24
	Total	**132**
Protocole de K. Lounis et M. Zulkernine [LZ21]	Authentificateur	112
	Objet connecté	124
	Total	**236**
Mbed TLS 1.3 avec authentification symétrique [RTB20]	**Total**	**381**
Mbed TLS 1.3 avec authentification asymétrique [RTB20]	**Total**	**1371**

TABLE 5.2 – Coût de communication de différents protocoles d'authentification.

Nous pouvons constater d'une part que le coût de communication de notre protocole est aligné avec le coût de protocoles similaires, voire plus faible en comparant aux protocoles de Aysu ou de Lounis et Zulkernine. D'autre part, l'ensemble des protocoles PUF possède un coût de communication plus faible que le protocole TLS dans sa version symétrique qui est la plus légère. Le protocole TLS dans sa version asymétrique affiche un coût supérieur d'un ordre de grandeur à celui des protocoles PUF. Ces résultats illustrent bien le potentiel des protocoles basés sur la technologie PUF pour réaliser de l'authentification sur des objets contraints.

Pour l'intégration de notre protocole d'authentification PUF dans le protocole TLS, nous avons également évalué le coût de communication en nous concentrant sur le *Handshake Protocol* de TLS. Nous avons, dans un premier temps, relevé la taille totale des données envoyées par l'objet connecté et l'authentificateur durant le *Handshake*. Nous avons répété cette procédure pour les algorithmes d'échange de clés classiques de TLS, à savoir : PSK, RSA et ECC. Nous avons choisi pour ces autres algorithmes des paramètres permettant d'avoir un niveau de sécurité équivalent, en nous appuyant sur les recommandations d'organismes tels que l'ANSSI [ANS20] ou le NIST [Bar20]. Ainsi, nous avons comparé notre protocole PUF-TLS manipulant des CRP de 128 bits avec les algorithmes PSK utilisant un secret de 128 bits, RSA utilisant une clé de

3072 bits et ECC utilisant une clé de 256 bits. Nous avons également comparé les algorithmes dans un scénario d'authentification mutuelle ; car c'est ce que propose notre protocole d'authentification. Les résultats sont présentés par le graphique de la figure 5.5.

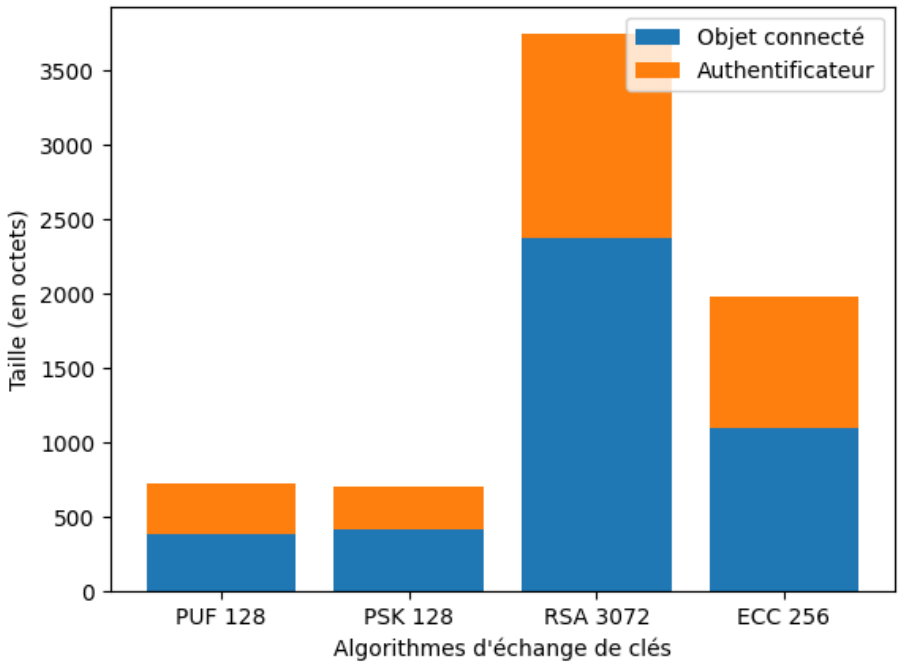

FIGURE 5.5 – Tailles des données échangés entre l'objet connecté et l'authentificateur selon l'algorithme d'échange de clés utilisé.

En observant ces résultats, nous constatons tout d'abord la faible empreinte des algorithmes PUF et PSK en termes de données échangées par rapport aux algorithmes RSA et ECC : l'algorithme RSA échange au total 5 fois plus de données que l'algorithme PUF, et l'algorithme ECC 2,7 fois plus. Mais, nous constatons également un déséquilibre entre la taille des données envoyées par l'objet connecté et la taille des données envoyées par l'authentificateur : quel que-soit l'algorithme, l'objet connecté envoie toujours plus de données que l'authentificateur. Cependant, ce déséquilibre est variable selon l'algorithme. Ainsi avec l'algorithme PUF, l'objet émet 52% du total des données mais avec l'algorithme RSA cela passe à 63%. Cela peut entraîner des coûts énergétiques d'émission plus importants pour l'objet connecté, ce qui peut diminuer sa durée de vie s'il s'agit d'un appareil fonctionnant sur batterie par exemple.

Pour affiner notre vision de ce déséquilibre, nous avons réalisé une seconde étude de la taille des données échangées en les classant cette fois-ci selon les quatre parties du *Handshake Protocol*, que nous avons présenté dans la section 4.4.3. Nous nous sommes concentrés dans cette seconde étude uniquement sur les données échangées au niveau du protocole TLS et nous avons fait abstractions des données liées aux en-têtes des protocoles des couches inférieures tels que TCP ou IP. Les résultats de cette seconde étude sont présentés par le graphique de la figure 5.6.

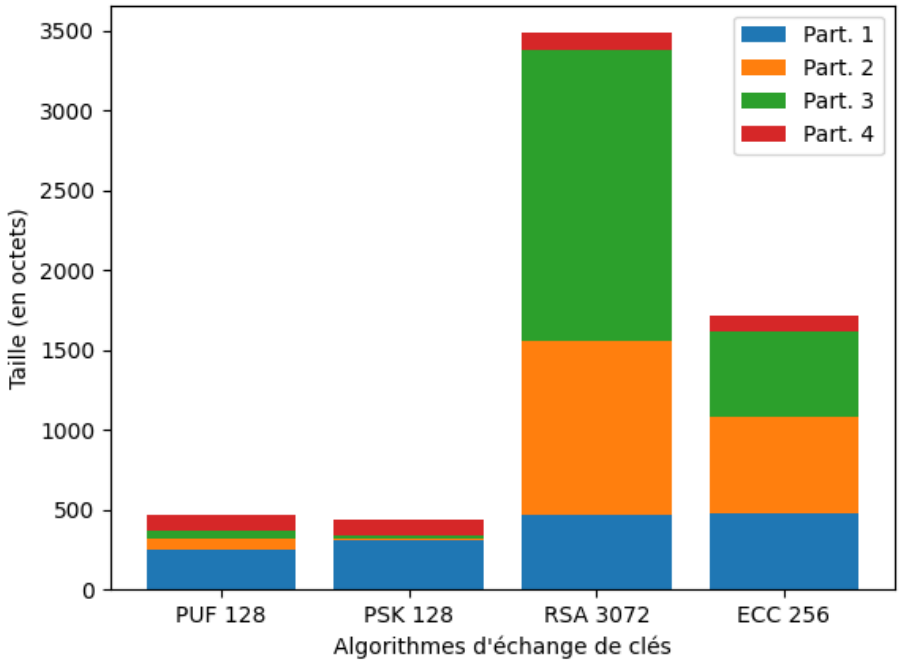

FIGURE 5.6 – Taille des données échangées par partie du Handshake Protocol de TLS, selon l'algorithme d'échange de clés utilisé.

Pour rappel, l'objet et l'authentificateur envoient chacun des données dans les parties 1 et 4 du protocole. Dans la partie 2, seul l'authentificateur émet des données ; et dans la partie 3, c'est l'objet qui est le seul à émettre. L'observation de ces résultats montre que, dans le cas du protocole RSA, c'est la partie 3 qui semble être à l'origine du déséquilibre puisqu'elle représente 52% du total des données échangées. Durant cette partie, l'objet connecté va non seulement transmettre son certificat, mais également un secret partagé ainsi qu'une signature des messages précédemment envoyés. C'est le cas également avec l'algorithme ECC, mais avec des tailles réduites ce qui conduit à une partie 3 représentant 31% du total des données échangées. Sur les algorithmes PUF et PSK n'utilisant pas de certificat, les parties 2 et 3 sont très réduites ce qui participe à leur faible empreinte. Il faut également noter que dans notre environnement de test, l'objet et l'authentificateur transmettent chacun un seul certificat. Il s'agit d'une simplification ; car, dans un déploiement concret, il est probable de trouver une chaîne de certification composée de plusieurs certificats. Dans ce cas, il peut être nécessaire d'envoyer plus d'un certificat ce qui augmente d'autant le coût de communication. Cela ne concerne que les algorithmes RSA et ECC.

5.2.3.4 *Métriques temporelles*

Nous avons également mesuré le temps nécessaire à la complétion de nos protocoles. Pour le protocole PUF, nous avons pu évaluer le temps d'exécution de chaque échange du protocole. Par échange, nous entendons une requête émise par la passerelle ou le registre avec la réponse associée de la part de l'objet. Pour mesurer ces durées, nous avons instrumenté le logiciel Python

implémentant le comportement de la passerelle dans le démonstrateur de la figure 5.2. Les temps moyens d'exécution sont présentés dans le tableau 5.3.

Etape	**Echange**	**Temps moyen (s)**
Enregistrement	INIT/RESP	1,1
	CHALL/RESP	3,0 * 10-3
Authentification	ID_REQ/ID_ANS	5,5 * 10-1
	AUTH/AUTH	1,1
Rechargement sécurisé	REFILL_AUTH/ REFILL_AUTH	1,1
	CHALL/RESP	6,4 * 10-3

TABLE 5.3 – Métriques temporelles du protocole PUF.

Avec ces résultats, nous avons constaté que le premier échange de l'étape d'enregistrement prenait un temps significativement plus long que les échanges suivants. Nous avons déterminé que cette différence provenait de la procédure d'initialisation, qui prend place lors de cet échange. Cette initialisation consiste en l'activation de la SRAM PUF, la récupération de la clé secrète stockée par la PUF et sa transmission au module AES. Après ces étapes seulement le challenge peut être traité par le module AES, qui le chiffre pour produire la réponse qui est renvoyée au registre. Ce travail d'initialisation est réalisé une seule fois, le module AES restant configuré pour les échanges suivants. Cela explique le faible temps d'exécution des échanges suivants.

Pour l'étape d'authentification, l'échange AUTH/AUTH demande une étape d'initialisation similaire à l'échange INIT/RESP, d'où le temps d'exécution similaire. A la fin de l'étape d'authentification, la SRAM PUF est désactivée et la clé secrète associée est effacée du module AES par sécurité. Cela explique ce temps d'exécution stable à travers les différentes authentifications. L'échange ID_REQ/ID_ANS est, quant à lui, plus simple ; car il nécessite simplement la reconstruction de l'identifiant de l'objet.

Concernant l'étape de rechargement sécurisé, nous constatons que l'authentification mutuelle de cette étape est complétée dans un temps similaire à l'étape d'authentification classique. Cela est positif ; mais est à nuancer du fait de l'utilisation d'un accélérateur cryptographique dans notre démonstrateur. Sur une cible ne disposant pas de ce type de périphérique, il est possible que le temps d'exécution soit rallongé par rapport à l'authentification classique. Nous constatons également que la génération de nouveaux CRP est plus lente que lors de l'étape d'enregistrement. Cela s'explique par l'ajout des étapes de chiffrement et de déchiffrement pour garantir la confidentialité des échanges.

Avec ces données temporelles, notamment celles concernant les étapes d'enregistrement et de rechargement sécurisé, nous nous sommes intéressés au temps total d'enregistrement d'un objet. Par temps total d'enregistrement, nous entendons le temps nécessaire à la génération d'un nombre suffisant de CRP pour couvrir la durée de vie entière d'un objet connecté. Cela demande en premier lieu d'évaluer combien de CRP sont nécessaires pour couvrir cette durée de vie. Ce nombre peut être hautement variable en fonction des applications, c'est

pourquoi nous avons choisi de nous placer dans le scénario d'un objet connecté ayant un rythme d'une authentification par minute pendant toute sa durée de vie. Nous avons réalisé notre évaluation pour trois valeurs de durée de vie parmi les plus courantes dans le domaine de l'IoT. Les résultats sont présentés dans le tableau 5.4.

Durée de vie objet (année)	**Nombre de CRP (millions de CRP)**	**Temps total d'enregistrement (h)**
1	2,1	1,8
5	11	8,8
10	21	18

TABLE 5.4 – Nombre de CRP et temps total d'enregistrement en fonction de la durée de vie d'un objet.

Ces résultats illustrent le défi représenté par la génération en une fois des données d'authentification pour toute la durée de vie d'un objet. Même pour une durée de vie d'un an, presque 2 heures sont nécessaires pour réaliser l'enregistrement d'un seul objet. Une telle durée n'est pas compatible avec un procédé industriel de fabrication. Cela implique également de disposer d'un registre d'authentification capable de sauvegarder plusieurs millions de CRP par objet. Selon le nombre d'objets produits, cela fait rapidement des milliards de CRP au total. La gestion de tels volumes pourrait demander des investissements conséquents de la part du fabricant ou du tiers de confiance en charge du registre. Enfin, au niveau des passerelles des clients, la simple gestion d'une dizaine d'objets demanderait le stockage de dizaines de millions de CRP. Bien que nous ayons supposé la passerelle moins contrainte que l'objet connecté, cela ne reste pas moins un équipement de terrain disposant de moins de ressources qu'un serveur dédié ou que des infrastructures *cloud*.

Au vu de ces constats, cette approche ne nous paraît pas réaliste pour l'authentification d'une flotte d'objets connectés.

C'est pourquoi nous avons proposé dans notre protocole une étape de rechargement qui permet de ne pas générer l'ensemble des données d'authentification lors de l'enregistrement. Seul un échantillon a besoin d'être généré, puis transmis à la passerelle adéquate, qui se chargera de réaliser les authentifications et les rechargements sécurisés au fil du temps, lorsque cela est nécessaire. La conséquence principale de cette nouvelle approche est la réduction du temps d'enregistrement pour s'aligner sur les contraintes industrielles des lignes de production. Pour concrétiser ce bénéfice, nous avons réalisé de nouvelles simulations du temps d'enregistrement en considérant des ensembles de CRP restreints, avec la possibilité de renouveler ces ensembles grâce au rechargement sécurisé. Nous avons également estimé la durée de vie de ces ensembles en restant sur le scénario précédent d'une authentification par minute. Les résultats sont présentés dans le tableau 5.5.

Nombre de CRP	Temps d'enregistrement (s)	Temps de rechargement sécurisé (s)	Durée de vie (d)
1440	5,4	10	1
10 * 10^3	31	65	7
43 * 10^3	131	277	30

TABLE 5.5 – Temps d'enregistrement, de rechargement sécurisé et durée de vie de l'authentification en fonction du nombre de CRP générés.

Avec cette nouvelle approche, il est par exemple possible de compléter l'étape d'enregistrement en moins d'une minute en ayant générer assez de données d'authentification pour une semaine d'utilisation. Cela nous paraît compatible avec un procédé industriel de fabrication de cartes électroniques. Une fois ces données initiales utilisées, la passerelle utilisera l'étape de rechargement sécurisé pour générer un nouvel ensemble de CRP de la taille souhaitée. Cela demandera la mise en pause de l'objet concerné le temps de la procédure ; environ 10 secondes pour générer des données pour un jour d'authentification et moins de 5 minutes pour un mois de durée de vie, ce qui nous semble raisonnable dans le cadre d'applications pervasives non critiques destinées à des maisons connectées. Selon les cas d'usage, ces périodes de rechargement peuvent même être synchronisées avec des périodes d'inactivité de l'environnement pervasif pour limiter leur impact. Dans notre exemple d'application pervasive de gestion de la température, cela peut être la nuit lorsque les usagers dorment ou bien lorsque la maison est vide.

Un autre avantage de cette approche est qu'elle permet à l'acquéreur de l'objet de fonctionner de manière autonome une fois les données d'authentification initiales récupérées. En effet, une fois passé cet échange, la passerelle n'a plus besoin de contacter le registre d'authentification. Cela peut être intéressant dans certaines configurations, tel qu'un réseau cloisonné et non connecté à Internet. Dans ce type de réseau, la transmission des données d'authentification initiales peut s'effectuer via un support physique, une clé USB par exemple, puis la passerelle sera en mesure de générer de manière autonome les CRP suivants pour assurer une authentification sur le long terme.

Pour résumer, nous avons implémenté avec succès deux démonstrateurs pour illustrer les fonctionnalités des protocoles sécurisés que nous avons conçu, basés sur la technologie PUF. Nous avons également évalué les performances de ces implémentations par rapports à d'autres propositions académiques ainsi que des solutions classiques. Nous avons montré que notre protocole d'authentification est compatible avec des objets contraints et que son intégration dans le protocole TLS réduit le coût de communication de l'étape d'authentification, bien que du travail supplémentaire soit nécessaire pour réduire l'empreinte mémoire du prototype.

Pour montrer la synergie entre nos deux contributions, la plateforme pervasive avec son extension de sécurité et le protocole de communication sécurisé utilisant la technologie PUF, nous avons intégrer le protocole dans la plateforme et nous l'avons représenté avec notre modèle de données pour évaluer

sa sécurité et permettre son utilisation par les applications pervasives. Nous présentons ces travaux dans la section suivante.

5.3 ÉVALUATION DE LA SÉCURITÉ DES OBJETS CONNECTÉS

Le démonstrateur de plateforme, que nous avons réalisé et présenté dans la section 5.1 nous a permis d'implémenter le modèle de données introduit dans la section 4.3.3. Ce modèle permet, notamment, de représenter les technologies de sécurité logicielles et matérielles utilisées par un objet géré par la plateforme.

Pour tester ce modèle, nous nous sommes placés dans un scénario pervasif illustré par la figure 5.7.

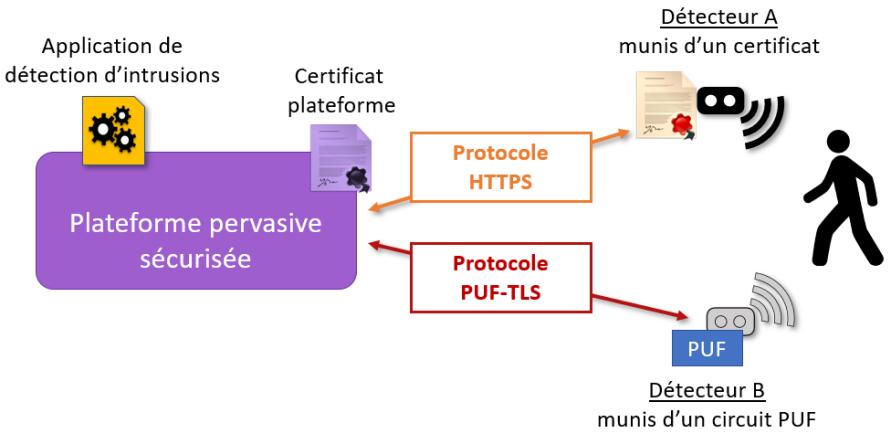

FIGURE 5.7 – Un exemple de scénario pervasif : une application de détection d'intrusion.

Dans ce scénario, une application pervasive de détection d'intrusion cherche à utiliser des détecteurs de présence répartis dans une habitation, nous en avons considéré deux dans notre exemple. Ces capteurs proviennent de fournisseurs différents et ils n'utilisent pas les mêmes technologies de sécurisation : l'un des capteurs (détecteur A) est livré avec un certificat et utilise le protocole HTTPS pour communiquer, tandis que l'autre capteur (détecteur B) possède un circuit PUF et utilise notre protocole PUF-TLS. Toutefois, ils fournissent le même service de détection, appelé *detectPresence*. Notons également que la plateforme dispose de son propre certificat pour permettre son authentification. Nous allons voir comment notre plateforme et son modèle de données permettent de rendre la sécurité plus lisible pour les applications dans ce cas de figure.

Pour commencer, rappelons que notre plateforme pervasive est capable de gérer l'hétérogénéité des protocoles de communication des objets connectés grâce à ses sous-modules de communication dédiés : les sous-modules HTTPS et PUF-TLS dans notre exemple. Ces sous-modules sont également capables, lors de l'enregistrement d'un nouvel objet, de détecter en partie les technologies de sécurité utilisées lors de la communication et de créer une instance de notre modèle associée à ce nouvel objet. La qualité de cette détection dépend, notamment, des informations mises à disposition par les composants logiciels utilisés par le sous-module pour gérer la communication. La figure 5.8 présente l'instance du modèle créée lors de l'enregistrement du détecteur A, communiquant en HTTPS, avec un focus sur la partie authentification, qui concerne principalement notre exemple.

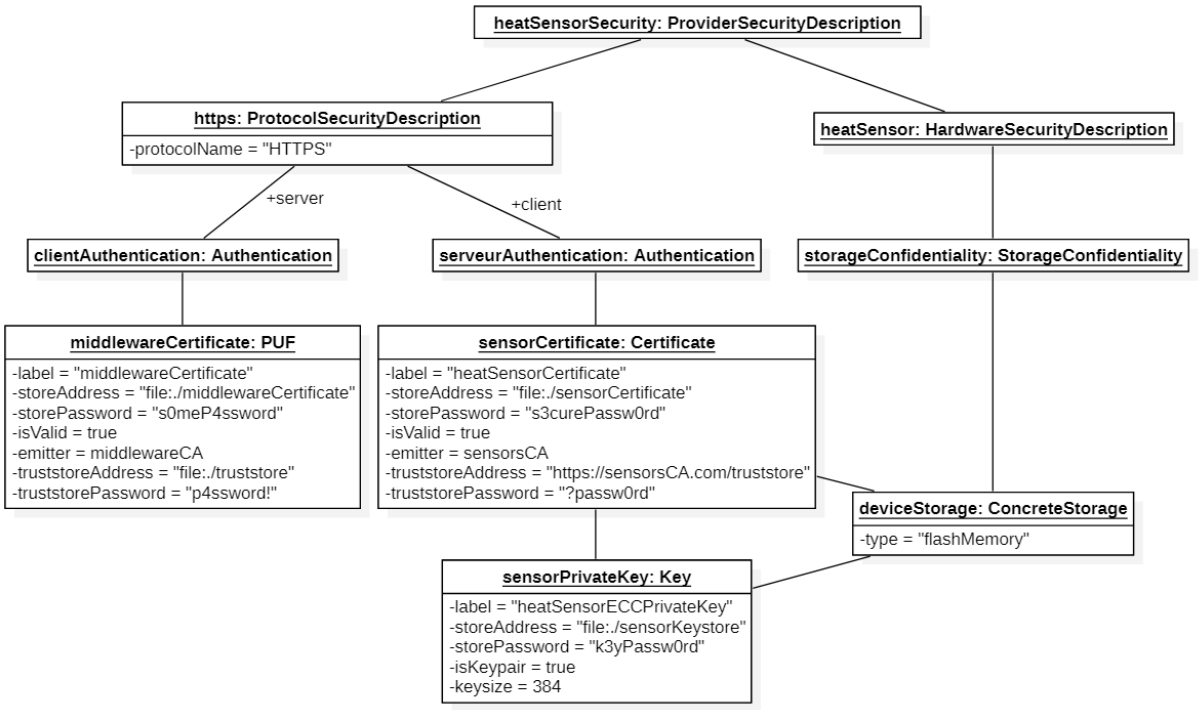

FIGURE 5.8 – Diagramme d'objets du détecteur A - partie authentification.

Ce diagramme détaille les technologies de sécurité utilisées pour permettre l'authentification de l'objet et de la plateforme. Il renseigne sur les élements d'authentification manipulées, tels que les certificats, ainsi que comment sont stockées ces élements. Cela permet de faire le lien entre la sécurité protocolaire et la sécurité matérielle des objets connectés. Par exemple, nous pouvons constater que le certificat du détecteur A et la clé privée qui lui est associée sont stockés dans une simple mémoire Flash, une mémoire non-volatile sans protection. Or, la confidentialité et l'intégrité des données offertes par le protocole HTTPS reposent sur cette clé privée. Le niveau de confiance dans cet objet peut donc est moindre par rapport à un objet utilisant une méthode de stockage sécurisée que nous avons présenté dans la section 3.4.

Le sous-module PUF-TLS génère également un diagramme d'objets lors de l'enregistrement du détecteur B, dont la partie dédiée à l'authentification est illustrée la par figure 5.9.

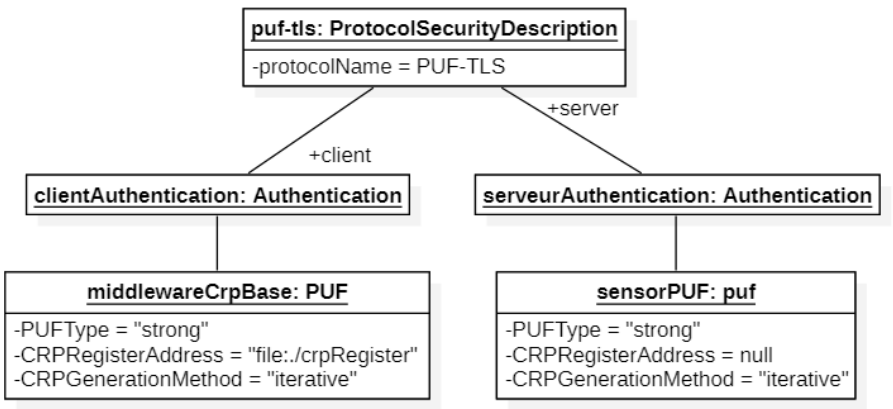

FIGURE 5.9 – Diagramme d'objets du détecteur B - partie authentification.

L'utilisation de la technologie PUF pour réaliser l'authentification mutuelle permet d'éviter le stockage d'un quelconque secret en mémoire, comme nous l'avons montré dans la section 4.4.

En tant que telles, ces informations sont peu exploitables par une application pervasive. Elles sont utilisées en interne de la plateforme pour instancier la partie abstraite de la description de la sécurité. Cette description abstraite indique les caractéristiques de sécurité haut niveau assurée par le fournisseur du service, c'est-à-dire l'objet connecté, et un score de confiance qui évalue la qualité des technologies de sécurité utilisées. Nous avons développé un premier algorithme de calcul du score de confiance, en prenant en compte la cohérence entre le niveau de sécurité d'un secret et le niveau de sécurité de sons sytème de stockage. Par exemple, dans le cas du détecteur A, une clé privée ECC de 384 bits pour réaliser de l'authentification offre normalement un niveau de sécurité fort, correspondant à un score de 75 (en prenant en compte les algorithmes de confidentialité et d'intégrité, non présentés sur les figures), mais son stockage dans une mémoire non protégée conduit notre algorithme d'évaluation à dégrader cette note à 50. L'utilisation d'un circuit PUF pour l'authentification est reconnue par notre algorithme comme offrant un niveau de sécurité modéré, avec un score de 60, en accord avec nos travaux. Bien sûr, il s'agit là d'un algorithme possible et il est possible de développer son propre algorithme d'évaluation pour s'adapter aux besoins et aux spécificités de domaines particuliers.

Lorsqu'une application fait une recherche de service auprès du **module Registre**, la description de chaque service est accompagnée de la ou des descriptions abstraites de sécurité correspondantes. La figure 5.10 illustre, dans notre exemple, le résultat d'une recherche de services par l'application pervasive au format JSON [10], un format de données communément utilisé par les API REST.

10. https://www.json.org/json-fr.html

```
[
  {
    "serviceName": "detectPresence",
    [...]
    "securityProfiles":
    [
      {
        "id": "a110[...]8453"'
        "securityLevel": 60,
        "providerAuthentication": true,
        "confidentiality": true,
        [...]
      },
      {
        "id": "9ea4[...]bb74",
        "securityLevel": 50,
        "providerAuthentication": true,
        "confidentiality": true,
        [...]
      }
    ]
  }
]
```

FIGURE 5.10 – Exemple de description de service avec les descriptions de sécurité associées.

Nous retrouvons le service *detectPresence* offert par les détecteurs de présence, avec deux descriptions de sécurité correspondant aux deux détecteurs. L'application peut utiliser ces informations pour choisir son service avec le niveau de sécurité qui lui convient, voire utiliser plusieurs niveaux de sécurité et combiner les résultats obtenus en fonction de ces niveaux, cela sans avoir à se soucier des technologies de sécurité utilisées de manière sous-jacente.

5.4 SYNTHÈSE

Dans ce chapitre, nous avons présenté pour chacune des deux contributions principales de cette thèse notre démarche pour la réalisation de démonstrateurs et pour l'évaluation de la contribution.

Pour notre plateforme pervasive et son extension de sécurité, nous avons mis en avant les avantages de l'utilisation de la plateforme en termes de simplification du développement d'une application pervasive faisant usage d'objets connectés hétérogènes en protocoles de communication et en sécurité, par rapport à un développement sans plateforme. L'utilisation de la plateforme simplifie non seulement le développement initial de l'application, mais également son évolution pour supporter de nouveaux objets et de nouveaux protocoles de communication.

Pour notre protocole de communication sécurisé et léger, basé sur la technologie PUF, nous avons présenté le développement de nos démonstrateurs pour la version de base du protocole permettant d'effectuer de l'authentification mu-

tuelle, puis pour une version plus avancée permettant, après l'authentification mutuelle des participants, d'établir une communication confidentielle et intègre, en utilisant les fonctionnalités du protocole de communication sécurisé TLS. Nous avons ensuite évalué la sécurité de notre protocole et ses performances en termes d'empreinte mémoire, de temps d'exécution et de taille des messages échangés. Nous avons pu valider la pertinence de notre contribution en comparant ces performances avec celles d'autres protocoles similaires. Enfin, nous avons mis en avant l'intérêt de l'étape de rechargement sécurisé de notre protocole pour réduire le temps d'enregistrement initial des objets et faciliter l'industrialisation de cette étape clé.

6 CONCLUSION ET PERSPECTIVES

6.1 SYNTHÈSE DES TRAVAUX

L'informatique pervasive promet une utilisation simple, quasiment naturelle, de services informatiques intégrés à nos environnements de vie et de travail : dans nos habitations, dans nos bureaux ou dans nos usines par exemple. Ces services facilitent la vie des utilisateurs en automatisant certaines tâches ou en simplifiant le travail collaboratif entre plusieurs utilisateurs. La diffusion toujours plus large d'objets connectés à nos côtés et les applications qui utilisent ces objets sont les fondations de ce concept d'informatique pervasive. Du fait de la nature presque invisible des solutions pervasives et de leur intégration dans de nombreux environnements, leur sécurisation est un enjeu majeur. Il s'agit toutefois d'un sujet complexe, pour plusieurs raisons.

Tout d'abord, les solutions de sécurité traditionnelles, largement déployées dans les systèmes informatiques, sont peu adaptées à la sécurisation des objets connectés intégrés aux solutions pervasives, à cause de la grande quantité d'objets à sécuriser et des ressources souvent limitées de ces objets. En pratique, diverses solutions de sécurité sont utilisées par les objets connectés aux niveaux protocolaire et matériel. Cela engendre une hétérogénéité des configurations de sécurité possibles qui complexifie l'utilisation des objets par les applications pervasives. Enfin, toutes les solutions de sécurité n'offrent pas la même qualité de protection. Déterminer le niveau de sécurité offert par une configuration de sécurité donnée est un travail complexe nécessitant une expertise dans le domaine. Pourtant, l'accès à ce type d'information nous paraît essentiel pour permettre aux applications pervasives de choisir les objets les plus sécurisés pour une tâche critique, ou pour qu'elles puissent adapter leur traitement lors de l'utilisation d'objets plus faiblement sécurisés.

Dans cette thèse, nous avions pour but de faciliter l'usage sécurisé d'objets connectés hétérogènes par des applications pervasives pour aboutir à des solutions pervasives sécurisées et de confiance. Aujourd'hui, des solutions basées sur des plateformes intermédiaires existent pour simplifier l'utilisation des objets connectés. Toutefois nous avons constaté, concernant la sécurité des objets connectés, que la plupart de ces plateformes :

— **remontent aux applications trop peu d'informations sur la sécurité des objets connectés**, ce qui ne permet pas aux applications de choisir des objets selon leur niveau de sécurité et qui conduit à l'utilisation non-discriminée d'objets ayant des niveaux de sécurité très différents.
— **se limitent au support des solutions de sécurité traditionnelles** et, de ce fait, ne permettent pas le support des objets connectés ne pouvant utiliser ces solutions.

Pour apporter des solutions à ces limitations, nous proposons une approche permettant de modéliser précisément la sécurité des objets connectés pour mieux la maîtriser et pour pouvoir intégrer une large gamme d'objets connectés

aux caractéristiques hétérogènes. Comme présenté dans le chapitre 4, notre approche prend la forme d'une plateforme pervasive sécurisée, qui permet le déploiement d'applications et simplifie l'utilisation d'objets connectés aux technologies de communication et de sécurité hétérogènes.

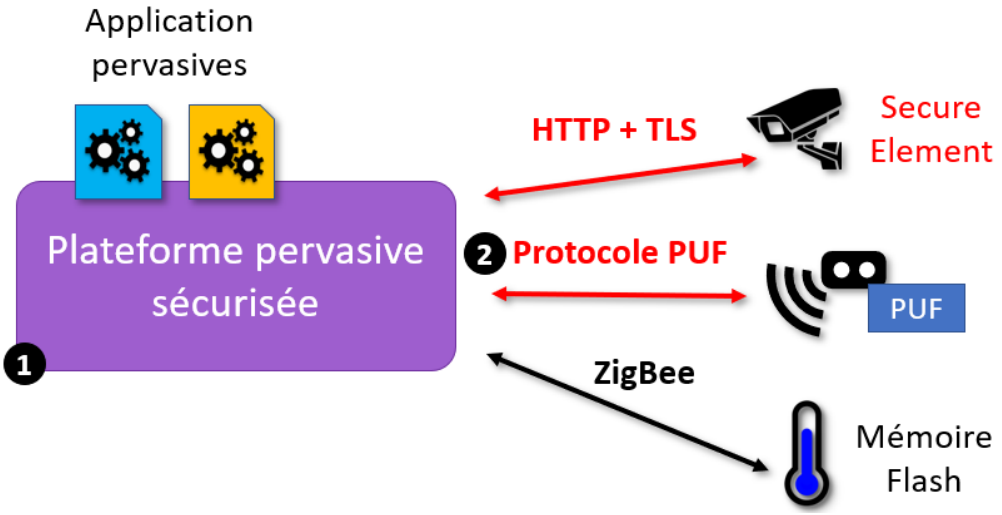

FIGURE 6.1 – Vue globale de notre approche

La mise en place de notre approche s'appuie sur deux éléments principaux :

— une **extension de plateforme pervasive dédiée à la sécurité (1)**, qui permet de gérer la communication sécurisée avec les objets connectés et de modéliser différentes configurations de sécurité.

— un **nouveau protocole de communication sécurisé et léger (2),** basé sur les fonctions physiques non-clonables ou PUF, pour simplifier la sécurisation des objets connectés contraints en ressources matérielles et en coûts, qui ne peuvent pas utiliser les solutions de sécurité traditionnelles.

Notre extension se base une plateforme pervasive utilisant l'approche à services pour masquer l'hétérogénéité des objets connectés et faciliter leur utilisation par les applications pervasives. Nous ajoutons à cette architecture de base la gestion de la sécurité des protocoles de communication en utilisant une approche modulaire : chaque protocole supporté dispose d'un module jouant le rôle d'un proxy, qui gére les différentes options de sécurité du protocole et son interfaçage avec la plateforme. Cette approche par modules facilite l'évolutivité et l'adaptabilité de la gestion des protocoles. L'extension comprend également des outils permettant de modéliser de manière précise les technologies de sécurité protocolaires et matérielles déployées sur un objet connecté et ainsi de produire une synthèse haut niveau de ces informations pour qu'elles soient utilisables facilement, c'est-à-dire sans un niveau d'expertise avancée en sécurité, par les applications pervasives dans le choix de leurs services.

Le protocole de communication que nous avons conçu permet de réaliser une authentification mutuelle entre un objet connecté et un équipement de supervision, en utilisant la technologie PUF pour générer et stocker, de manière à la fois sécurisée et à bas coût, une empreinte unique par objet. Ce protocole a un coût de déploiement faible et il est peu consommateur de ressources embarquées par rapport aux solutions d'authentification traditionnelles, telles

que l'authentification avec des certificats et des Infrastructures à Clés Publiques. De plus, nous avons utilisé une approche novatrice pour la gestion des données d'authentification, en permettant la génération de nouvelles données de manière sécurisée lorsque l'objet est déployé sur le terrain. Cela permet de réduire significativement le volume de données à gérer par objet, ce qui facilite le passage à l'échelle de la solution. Dans sa version de base, le protocole réalise uniquement de l'authentification. Nous avons également conçu une version permettant de réaliser de l'échange de secrets authentifié et nous avons intégré cette version au protocole sécurisé TLS comme alternative moins consommatrice de ressources et plus facile à passer à l'échelle que les solutions traditionnelles présentes dans le protocole, telles que l'authentification par certificat ou l'authentification via un secret partagé.

Notre approche a été validée par la réalisation de plusieurs démonstrateurs implémentant les contributions précédemment présentées : un démonstrateur de plateforme pervasive avec notre extension de sécurité, un démonstrateur de l'authentification avec notre protocole PUF ainsi qu'un démonstrateur d'une communication sécurisée avec le protocole TLS et une authentification par PUF. Ces démonstrateurs ont illustré comment une application pervasive pouvait choisir entre plusieurs services offerts par des objets connectés en s'appuyant sur les caractéristiques de sécurité liées à la configuration de chaque objet et comment il était possible de sécuriser un objet connecté contraint avec la technologie PUF. Nous avons également pu montrer la synergie de nos contributions, en concevant pour le démonstrateur de la plateforme un module prenant en charge le protocole TLS avec authentification PUF. Les résultats expérimentaux de notre protocole ont permis de valider ses performances et sa faible consommation de ressources par rapport aux solutions traditionnelles.

6.2 PERSPECTIVES

Les contributions présentées dans ce manuscrit apportent des éléments de réponse permettant de sécuriser les futures solutions pervasives. Toutefois, de nombreuses questions restent ouvertes dans le domaine de la sécurisation des solutions pervasives que nous avons abordées. Dans cette dernière section, nous partageons des trajectoires qui nous paraissent pertinentes pour de futurs travaux de recherche dans le domaine.

La conception de nouveaux algorithmes d'évaluation de la sécurité des objets connectés. Nous avons proposé dans notre architecture de plateforme pervasive sécurisée un premier algorithme d'évaluation de la sécurité d'un objet connecté en fonction des technologies qu'il utilise, à la fois au niveau protocolaire et au niveau matériel. Toutefois, cet algorithme est, en l'état, une simple preuve de concept et nous pensons qu'il peut être étoffé pour prendre en compte plus de facteurs, comme la cohérence dans le choix des algorithmes de chiffrement et de contrôle d'intégrité, ou la présence de vulnérabilités connues dans les algorithmes utilisés. Ces algorithmes pourraient être intégrés à notre plateforme, mais nous avions également envisagé la possibilité que ces algorithmes soient gérés par des organismes tiers, dédiés à l'évaluation des modèles de sécurité. Nous pensons que cette piste d'externalisation et les implications associées méritent également une étude plus approfondie.

Le déploiement d'outils d'intelligence artificielle au service de la sécurisation des environnements pervasifs. Nous vivons actuellement une forte effervescence du domaine de l'intelligence artificielle, qui s'accompagne du développement et du déploiement d'outils pour diverses finalités : le traitement d'images, la synthèse de contenu ou l'aide à la décision, par exemple. Le domaine de la sécurité informatique est également concerné, avec l'arrivée d'outils aux objectifs variés, allant de l'analyse de configuration d'un système pour mieux le sécuriser à la découverte de vecteurs d'attaque de manière plus efficace. Il nous paraît intéressant d'étudier comment ces nouveaux outils peuvent être utilisés pour participer à la sécurisation des futures solutions pervasives. Par exemple, l'architecture de plateforme sur laquelle nous nous sommes basés dans cette thèse pourrait être encore étendue pour permettre le déploiement d'outils basés sur l'intelligence artificielle. Il nous semble possible de développer un ou plusieurs nouveaux modules capable de collecter, stocker et analyser les données échangées en interne de la plateforme, comme l'enregistrement de nouveaux objets ou la remontée de mesures, grâce à l'approche microservices et la communication sous forme de messages de notre plateforme. Il faudrait toutefois étudier comment réaliser au mieux cette ou ces extensions, qu'elles pourraient être les attentes raisonnables et les limites de tels outils et quelles données utiliser pour fournir les meilleurs résultats, par exemple.

La mise en œuvre d'un marché de l'occasion sécurisé pour les objets connectés. L'Internet des Objets est un domaine en expansion continue et le nombre d'objets connectés mis en circulation ne cesse d'augmenter. L'informatique pervasive repose sur la forte diffusion d'objets connectés dans nos environnements pour les rendre connectés, donc utilisables par des applications pervasives. Cependant, cette croissance devra aussi se confronter aux problématiques d'impact écologique de la production des objets connectés et la finitude des ressources

nécessaire à cette production. Pour rendre le domaine plus vertueux, participer à la réduction de l'empreinte carbone des objets connectés et la réduction de leur consommation en ressources, aussi bien en matières premières qu'en ressources énergétiques, développer un marché de l'occasion des objets connectés nous semble une idée prometteuse. Toutefois, cela soulève de nombreuses questions, en termes de sécurité notamment. Il nous paraît important de réfléchir à la gestion des secrets chargés dans les objets ou encore aux procédures de traitement des objets à leur entrée et à leur sortie de ce marché de l'occasion. Il nous semble également intéressant d'étudier la compatibilité des protocoles et des technologies existantes de l'IoT à une seconde vie pour un objet connecté. Par exemple, plusieurs protocoles d'authentification basés sur la technologie PUF, comme celui que nous avons conçu et présenté, sont basés sur l'utilisation unique de données issues des circuits PUF. Se posent alors des questions sur la gestion *Challenge-Response Pairs* non utilisées lors du déploiement initial de l'objet, pour les désactiver et s'assurer qu'un ancien propriétaire d'objet ne puisse pas en reprendre le contrôle une fois qu'il s'en est séparé, ou encore sur la remise à zéro de certains paramètres du protocle, le compteur anti-rejeu dans le cas de notre contribution, et l'impact potentiel sur la sécurité. Pour répondre à ces problématique, il est possible de considérer des architectures de PUF paramétrables, avec un ou plusieurs paramètres liés au propriétaire actuel ou à l'environnement de déploiement courant, et comment adapter, si cela est possible, des protocoles d'authentification comme le notre à ce nouveau type de PUF.

Revue

Arthur Desuert, Stéphanie Chollet, Laurent Pion et David Hély. « Refillable PUF Authentication Protocol for Constrained Devices ». In : *Journal of Ambient Intelligence and Smart Environments* 14.3 (jan. 2022), p. 195-212. issn : 1876-1364. doi : 10.3233/AIS-210325.

Conférences

Arthur Desuert, Stéphanie Chollet, Laurent Pion et David Hély. « PUF-Based Protocol for Securing Constrained Devices ». In : *2021 17th International Conference on Intelligent Environments (IE)*. Juin 2021, p. 1-8. doi : 10.1109/IE51775.2021.9486492.

Arthur Desuert, Stéphanie Chollet, Laurent Pion et David Hély. « A Middleware for Secure Integration of Heterogeneous Edge Devices ». In : *2022 IEEE International Conference on Edge Computing and Communications (EDGE)*. Juill. 2022, p. 83-92. doi : 10.1109/EDGE55608.2022.00023.

Tutoriel

Arthur Desuert et Amir Ali Pour. « An Overview of the Security Challenges of IoT Solutions ». In : *2022 18th Internation Conference on Intelligent Environments (IE)*. Juin 2022

Poster

Arthur Desuert, Stéphanie Chollet, Laurent Pion et David Hély. « A Middleware for Secure Integration of Heterogeneous Edge Devices ». Presented at : *Journée du laboratoire LCIS*, LCIS, Juin 2022

BIBLIOGRAPHIE

[ANS20] ANSSI. *Guide des mécanismes cryptographiques - Règles et recommandations concernant le choix et le dimensionnement des mécanismes cryptographiques*. Jan. 2020. (Visité le 03/03/2023).

[AWSa] AWS. *AWS IoT Device Shadow Service - AWS IoT Core*. `https://docs.aws.amazon.com/iot/latest/developerguide/iot-device-shadows.html`. (Visité le 30/03/2021).

[AWSb] AWS. *Device Provisioning - AWS IoT Core*. `https://docs.aws.amazon.com/iot/latest/developerguide/iot-provision.html`. (Visité le 10/03/2021).

[AWSc] AWS. *Managing Devices with AWS IoT - AWS IoT Core*. `https://docs.aws.amazon.com/iot/latest/developerguide/iot-thing-management.html`. (Visité le 29/03/2021).

[AWSd] AWS. *Sécurité IoT | Gestion de la sécurité des appareils IoT | AWS IoT Device Defender*. `https://aws.amazon.com/fr/iot-device-defender/`. (Visité le 09/03/2021).

[And+19] Andrew Banks, Ed Briggs, Ken Borgendale et Rahul Gupta. *MQTT Version 5.0*. Mars 2019. (Visité le 05/10/2021).

[Ant+17] Manos Antonakakis et al. « Understanding the Mirai Botnet ». In : *26th {USENIX} Security Symposium ({USENIX} Security 17)*. 2017, p. 1093-1110. ISBN : 978-1-931971-40-9. (Visité le 10/06/2021).

[ATR20] Daniele Antonioli, Nils Ole Tippenhauer et Kasper Rasmussen. « Key Negotiation Downgrade Attacks on Bluetooth and Bluetooth Low Energy ». In : *ACM Transactions on Privacy and Security* 23.3 (juill. 2020), 14 :1-14 :28. ISSN : 2471-2566. DOI : 10.1145/3394497. (Visité le 26/05/2023).

[AS99] W. S. Ark et T. Selker. « A Look at Human Interaction with Pervasive Computers ». In : *IBM Systems Journal* 38.4 (1999), p. 504-507. ISSN : 0018-8670. DOI : 10.1147/sj.384.0504.

[Arm+11] Frederik Armknecht, Roel Maes, Ahmad-Reza Sadeghi, François-Xavier Standaert et Christian Wachsmann. « A Formalization of the Security Features of Physical Functions ». In : *2011 IEEE Symposium on Security and Privacy*. Mai 2011, p. 397-412. DOI : 10.1109/SP.2011.10.

[Aug83] Auguste Kerckhoffs. « La Cryptographie Militaire ». In : *Journal des sciences militaires* (1883).

[Avi+04] A. Avizienis, J.-C. Laprie, B. Randell et C. Landwehr. « Basic Concepts and Taxonomy of Dependable and Secure Computing ». In : *IEEE Transactions on Dependable and Secure Computing* 1.1 (jan. 2004), p. 11-33. ISSN : 1941-0018. DOI : 10.1109/TDSC.2004.2.

[Ayg17] Colin Aygalinc. « Un modèle à composant pour la gestion de contextes pervasifs orientés service ». Thèse de doct. Université Grenoble Alpes, déc. 2017. (Visité le 24/06/2021).

[Ays+15] Aydin Aysu, Ege Gulcan, Daisuke Moriyama, Patrick Schaumont et Moti Yung. « End-To-End Design of a PUF-Based Privacy Preserving Authentication Protocol ». In : *Cryptographic Hardware and Embedded Systems – CHES 2015*. Sous la dir. de Tim Güneysu et Helena Handschuh. T. 9293. Lecture Notes in Computer Science. Berlin, Heidelberg : Springer, sept. 2015, p. 556-576. ISBN : 978-3-662-48324-4. DOI : 10.1007/978-3-662-48324-4_28.

[Bar+19] Mario Barbareschi, Alessandra De Benedictis, Erasmo La Montagna, Antonino Mazzeo et Nicola Mazzocca. « PUF-Enabled Authentication-as-a-Service in Fog-IoT Systems ». In : *2019 IEEE 28th International Conference on Enabling Technologies : Infrastructure for Collaborative Enterprises (WETICE)*. Juin 2019, p. 58-63. DOI : 10.1109/WETICE.2019.00020.

[BDBM18] Mario Barbareschi, Alessandra De Benedictis et Nicola Mazzocca. « A PUF-based Hardware Mutual Authentication Protocol ». In : *Journal of Parallel and Distributed Computing* 119 (sept. 2018), p. 107-120. ISSN : 0743-7315. DOI : 10.1016/j.jpdc.2018.04.007. (Visité le 03/12/2020).

[Bar12] Jonathan Bardin. « RoSe : un framework pour la conception et l'exécution d'applications distribuées dynamiques et hétérogènes ». Thèse de doct. Université de Grenoble, oct. 2012. (Visité le 03/02/2021).

[Bar20] Elaine Barker. *Recommendation for Key Management : Part 1 – General*. Rapp. tech. NIST Special Publication (SP) 800-57 Part 1 Rev. 5. National Institute of Standards and Technology, mai 2020. DOI : 10.6028/NIST.SP.800-57pt1r5. (Visité le 11/06/2021).

[Bec+14] Michael Till Beck, Martin Werner, Sebastian Feld et S. Schimper. « Mobile Edge Computing : A Taxonomy ». In : *Proc. of the Sixth International Conference on Advances in Future Internet*. Citeseer, 2014, p. 48-55.

[Bec+04] C. Becker, M. Handte, G. Schiele et K. Rothermel. « PCOM - a Component System for Pervasive Computing ». In : *Second IEEE Annual Conference on Pervasive Computing and Communications, 2004. Proceedings of The*. Mars 2004, p. 67-76. DOI : 10.1109/PERCOM.2004.1276846.

[Bec+03] C. Becker, G. Schiele, H. Gubbels et K. Rothermel. « BASE - a Micro-Broker-Based Middleware for Pervasive Computing ». In : *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications, 2003. (PerCom 2003).* Mars 2003, p. 443-451. DOI : 10.1109/PERCOM.2003.1192769.

[Bec+19] Christian Becker, Christine Julien, Philippe Lalanda et Franco Zambonelli. « Pervasive Computing Middleware : Current Trends and Emerging Challenges ». In : *CCF Transactions on Pervasive Computing and Interaction* 1.1 (mai 2019), p. 10-23. ISSN : 2524-5228. DOI : 10.1007/s42486-019-00005-2. (Visité le 27/10/2020).

[Ber+14] Benjamin Bertran, Julien Bruneau, Damien Cassou, Nicolas Loriant, Emilie Balland et Charles Consel. « DiaSuite : A Tool Suite to Develop Sense/Compute/Control Applications ». In : *Science of Computer Programming*. Experimental Software and Toolkits (EST 4) : A Special Issue of the Workshop on Academic Software Development Tools and Techniques (WASDeTT-3 2010) 79 (jan. 2014), p. 39-51. ISSN : 0167-6423. DOI : 10.1016/j.scico.2012.04.001. (Visité le 08/03/2021).

[BM14] M. Bhargava et K. Mai. « An Efficient Reliable PUF-based Cryptographic Key Generator in 65nm CMOS ». In : *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*. Mars 2014, p. 1-6. DOI : 10.7873/DATE.2014.083.

[Bon+14] Flavio Bonomi, Rodolfo Milito, Preethi Natarajan et Jiang Zhu. « Fog Computing : A Platform for Internet of Things and Analytics ». In : *Big Data and Internet of Things : A Roadmap for Smart Environments*. Sous la dir. de Nik Bessis et Ciprian Dobre. Studies in Computational Intelligence. Cham : Springer International Publishing, 2014, p. 169-186. ISBN : 978-3-319-05029-4. DOI : 10.1007/978-3-319-05029-4_7. (Visité le 26/01/2023).

[Bon+12] Flavio Bonomi, Rodolfo Milito, Jiang Zhu et Sateesh Addepalli. « Fog Computing and Its Role in the Internet of Things ». In : *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*. MCC '12. New York, NY, USA : Association for Computing Machinery, août 2012, p. 13-16. ISBN : 978-1-4503-1519-7. DOI : 10.1145/2342509.2342513. (Visité le 26/01/2023).

[CMA14] C. Bormann, M. Ersue et A. Keranen. *Terminology for Constrained-Node Networks*. https://datatracker.ietf.org/doc/html/rfc7228. Mai 2014. (Visité le 23/06/2021).

[Cap+14] Mauro Caporuscio, Marco Funaro, Carlo Ghezzi et Valérie Issarny. « ubiREST : A RESTful Service-Oriented Middleware for Ubiquitous Networking ». In : *Advanced Web Services*. Sous la dir. d'Athman Bouguettaya, Quan Z. Sheng et Florian Daniel. New York, NY : Springer, 2014, p. 475-500. ISBN : 978-1-4614-7535-4. DOI : 10.1007/978-1-4614-7535-4_20. (Visité le 09/03/2022).

[CRI12] Mauro Caporuscio, Pierre-Guillaume Raverdy et Valerie Issarny. « ubiSOAP : A Service-Oriented Middleware for Ubiquitous Networking ». In : *IEEE Transactions on Services Computing* 5.1 (jan. 2012), p. 86-98. ISSN : 1939-1374. DOI : 10.1109/TSC.2010.60.

[CBC10] Damien Cassou, Julien Bruneau et Charles Consel. « A Tool Suite to Prototype Pervasive Computing Applications ». In : *2010 8th IEEE International Conference on Pervasive Computing and Commu-*

nications Workshops (PERCOM Workshops). Mars 2010, p. 820-822. DOI : 10.1109/PERCOMW.2010.5470550.

[Cay+21] Romain CAYRE, Florent GALTIER, Guillaume AURIOL, Vincent NICOMETTE, Mohamed KAÂNICHE et Géraldine MARCONATO. « InjectaBLE : Injecting Malicious Traffic into Established Bluetooth Low Energy Connections ». In : *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. Juin 2021, p. 388-399. DOI : 10.1109/DSN48987.2021.00050.

[Cho09] Stéphanie CHOLLET. « Orchestration de services hétérogènes et sécurisés ». Thèse de doct. Université Joseph-Fourier - Grenoble I, déc. 2009. (Visité le 21/10/2021).

[CL08] Stéphanie CHOLLET et Philippe LALANDA. « Security Specification at Process Level ». In : *2008 IEEE International Conference on Services Computing*. T. 1. Juill. 2008, p. 165-172. DOI : 10.1109/SCC.2008.51.

[Cis20] CISCO. *Cisco Annual Internet Report (2018–2023)*. Mars 2020. (Visité le 05/05/2021).

[Coo+08] D. COOPER, S. SANTESSON, S. FARRELL, S. BOEYEN, R. HOUSLEY et W. POLK. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. Rapp. tech. RFC5280. RFC Editor, mai 2008, RFC5280. DOI : 10.17487/rfc5280. (Visité le 13/12/2021).

[Des+21] Arthur DESUERT, Stéphanie CHOLLET, Laurent PION et David HÉLY. « PUF-Based Protocol for Securing Constrained Devices ». In : *2021 17th International Conference on Intelligent Environments (IE)*. Juin 2021, p. 1-8. DOI : 10.1109/IE51775.2021.9486492.

[Des+22a] Arthur DESUERT, Stéphanie CHOLLET, Laurent PION et David HÉLY. « A Middleware for Secure Integration of Heterogeneous Edge Devices ». In : *2022 IEEE International Conference on Edge Computing and Communications (EDGE)*. Juill. 2022, p. 83-92. DOI : 10.1109/EDGE55608.2022.00023.

[Des+22b] Arthur DESUERT, Stéphanie CHOLLET, Laurent PION et David HÉLY. « Refillable PUF Authentication Protocol for Constrained Devices ». In : *Journal of Ambient Intelligence and Smart Environments* 14.3 (jan. 2022), p. 195-212. ISSN : 1876-1364. DOI : 10.3233/AIS-210325. (Visité le 28/06/2022).

[Dey01] Anind K. DEY. « Understanding and Using Context ». In : *Personal and Ubiquitous Computing* 5.1 (fév. 2001), p. 4-7. ISSN : 1617-4909. DOI : 10.1007/s007790170019. (Visité le 02/09/2021).

[DY81] D. DOLEV et A. C. YAO. « On the Security of Public Key Protocols ». In : *22nd Annual Symposium on Foundations of Computer Science (Sfcs 1981)*. Oct. 1981, p. 350-357. DOI : 10.1109/SFCS.1981.32.

[DD17] Koustabh DOLUI et Soumya Kanti DATTA. « Comparison of Edge Computing Implementations : Fog Computing, Cloudlet and Mobile Edge Computing ». In : *2017 Global Internet of Things Summit (GIoTS)*. Juin 2017, p. 1-6. DOI : 10.1109/GIOTS.2017.8016213.

[Don11] Donald E. Eastlake 3rd. *Transport Layer Security (TLS) Extensions : Extension Definitions*. Request for Comments RFC 6066. Internet Engineering Task Force, jan. 2011. DOI : 10.17487/RFC6066. (Visité le 28/02/2023).

[Dor+15] Bruno Dorsemaine, Jean-Philippe Gaulier, Jean-Philippe Wary, Nizar Kheir et Pascal Urien. « Internet of Things : A Definition & Taxonomy ». In : *2015 9th International Conference on Next Generation Mobile Applications, Services and Technologies*. Sept. 2015, p. 72-77. DOI : 10.1109/NGMAST.2015.71.

[Dwo+01] Morris J. Dworkin, Elaine B. Barker, James R. Nechvatal, James Foti, Lawrence E. Bassham, E. Roback et James F. Dray Jr. « Advanced Encryption Standard (AES) ». In : (nov. 2001). (Visité le 13/06/2022).

[ERA10] Markus Eisenhauer, Peter Rosengren et Pablo Antolin. « HYDRA : A Development Platform for Integrating Wireless Devices and Sensors into Ambient Intelligence Systems ». In : *The Internet of Things*. Sous la dir. de Daniel Giusto, Antonio Iera, Giacomo Morabito et Luigi Atzori. New York, NY : Springer, 2010, p. 367-373. ISBN : 978-1-4419-1674-7. DOI : 10.1007/978-1-4419-1674-7_36.

[ECL14] C. Escoffier, S. Chollet et P. Lalanda. « Lessons Learned in Building Pervasive Platforms ». In : *2014 IEEE 11th Consumer Communications and Networking Conference (CCNC)*. Jan. 2014, p. 7-12. DOI : 10.1109/CCNC.2014.6866540.

[Far13] Aurélien Faravelon. « Une démarche de conception et d'implémentation de la protection de la vie privée basée sur le contrôle d'accès appliquée aux compositions de services ». Thèse de doct. Université de Grenoble, déc. 2013. (Visité le 19/11/2021).

[Far+21] Fadi Farha, Huansheng Ning, Karim Ali, Liming Chen et Christopher Nugent. « SRAM-PUF-Based Entities Authentication Scheme for Resource-Constrained IoT Devices ». In : *IEEE Internet of Things Journal* 8.7 (avr. 2021), p. 5904-5913. ISSN : 2327-4662. DOI : 10.1109/JIOT.2020.3032518.

[FR14] Roy T. Fielding et Julian Reschke. *Hypertext Transfer Protocol (HTTP/1.1) : Semantics and Content*. Request for Comments RFC 7231. Internet Engineering Task Force, juin 2014. DOI : 10.17487/RFC7231. (Visité le 05/10/2021).

[Gas+02] Blaise Gassend, Dwaine Clarke, Marten van Dijk et Srinivas Devadas. « Silicon Physical Random Functions ». In : *Proceedings of the 9th ACM Conference on Computer and Communications Security*. CCS '02. New York, NY, USA : Association for Computing Machinery, nov. 2002, p. 148-160. ISBN : 978-1-58113-612-8. DOI : 10.1145/586110.586132. (Visité le 17/12/2020).

[Ger+17] Eva Gerbert-Gaillard, Philippe Lalanda, Stéphanie Chollet et Jérémie Demarchez. « A Self-Aware Approach to Context Management in Pervasive Platforms ». In : *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. Mars 2017, p. 256-261. DOI : 10.1109/PERCOMW.2017.7917568.

[Glo15] Global Platform. *The Trusted Execution Environment : Delivering Enhanced Security at a Lower Cost to the Mobile Market*. https://globalplatform.org/wp-content/uploads/2018/04/GlobalPlatform_TEE_Whitepaper_2015.pdf. Juin 2015. (Visité le 20/11/2020).

[Gün14] Ozan Necati Günalp. « Continuous Deployment of Pervasive Applications in Dynamic Environments ». These de Doctorat. Grenoble, nov. 2014. (Visité le 02/09/2021).

[Had18] Rania Ben Hadj. « Gestion de conflits dans une plateforme ubiquitaire orientée services ». Thèse de doct. Université Grenoble Alpes, avr. 2018. (Visité le 25/04/2023).

[HA06] Munirul Haque et Sheikh Ahamed. « Security in Pervasive Computing : Current Status and Open Issues ». In : *International Journal of Network Security* (nov. 2006).

[HBF09] Daniel E. Holcomb, Wayne P. Burleson et Kevin Fu. « Power-Up SRAM State as an Identifying Fingerprint and Source of True Random Numbers ». In : *IEEE Transactions on Computers* 58.9 (sept. 2009), p. 1198-1210. ISSN : 0018-9340. DOI : 10.1109/TC.2008.212. (Visité le 10/03/2020).

[Int05] International Telecommunication Union. *ITU Internet Reports 2005 : The Internet of Things*. Rapp. tech. Nov. 2005. (Visité le 17/01/2023).

[JCL11] Henner Jakob, Charles Consel et Nicolas Loriant. « Architecturing Conflict Handling of Pervasive Computing Resources ». In : *Distributed Applications and Interoperable Systems*. Sous la dir. de Pascal Felber et Romain Rouvoy. Lecture Notes in Computer Science. Berlin, Heidelberg : Springer, 2011, p. 92-105. ISBN : 978-3-642-21387-8. DOI : 10.1007/978-3-642-21387-8_8.

[KMS95] P. Karn, P. Metzger et W. Simpson. *The ESP Triple DES Transform*. Rapp. tech. RFC1851. RFC Editor, sept. 1995, RFC1851. DOI : 10.17487/rfc1851. (Visité le 09/07/2023).

[Kim+18] Jeremie S. Kim, Minesh Patel, Hasan Hassan et Onur Mutlu. « The DRAM Latency PUF : Quickly Evaluating Physical Unclonable Functions by Exploiting the Latency-Reliability Tradeoff in Modern Commodity DRAM Devices ». In : *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. Fév. 2018, p. 194-207. DOI : 10.1109/HPCA.2018.00026.

[Kob87] Neal Koblitz. « Elliptic Curve Cryptosystems ». In : *Mathematics of Computation* 48.177 (1987), p. 203-209. ISSN : 0025-5718, 1088-6842. DOI : 10.1090/S0025-5718-1987-0866109-5. (Visité le 12/12/2022).

[KJJ99] Paul KOCHER, Joshua JAFFE et Benjamin JUN. « Differential Power Analysis ». In : *Advances in Cryptology — CRYPTO' 99*. Sous la dir. de Michael WIENER. Lecture Notes in Computer Science. Berlin, Heidelberg : Springer, 1999, p. 388-397. ISBN : 978-3-540-48405-9. DOI : 10.1007/3-540-48405-1_25.

[LK10] Hyun Jung LA et Soo Dong KIM. « A Service-Based Approach to Designing Cyber Physical Systems ». In : *2010 IEEE/ACIS 9th International Conference on Computer and Information Science*. Août 2010, p. 895-900. DOI : 10.1109/ICIS.2010.73.

[Lal+14] P. LALANDA, C. HAMON, C. ESCOFFIER et T. LEVEQUE. « iCasa, a Development and Simulation Environment for Pervasive Home Applications ». In : *2014 IEEE 11th Consumer Communications and Networking Conference (CCNC)*. Jan. 2014, p. 1142-1143. DOI : 10.1109/CCNC.2014.6940512.

[Lal23] Philippe LALANDA. « Edge Computing and Learning ». In : *The Evolution of Pervasive Information Systems*. Sous la dir. de Manuele KIRSCH PINHEIRO, Carine SOUVEYET, Philippe ROOSE et Luiz Angelo STEFFENEL. Cham : Springer International Publishing, 2023, p. 99-122. ISBN : 978-3-031-18175-7 978-3-031-18176-4. DOI : 10.1007/978-3-031-18176-4_5. (Visité le 28/06/2023).

[Lal+21] Philippe LALANDA, German VEGA, Humberto CERVANTES et Denis MORAND. « Architecture and Pervasive Platform for Machine Learning Services in Industry 4.0 ». In : *2021 IEEE International Conference on Pervasive Computing and Communications Workshops and Other Affiliated Events (PerCom Workshops)*. Mars 2021, p. 293-298. DOI : 10.1109/PerComWorkshops51409.2021.9431009.

[Lan+16] Adam LANGLEY, Wan-Teh CHANG, Nikos MAVROGIANNOPOULOS, Joachim STROMBERGSON et Simon JOSEFSSON. *ChaCha20-Poly1305 Cipher Suites for Transport Layer Security (TLS)*. Request for Comments RFC 7905. Internet Engineering Task Force, juin 2016. DOI : 10.17487/RFC7905. (Visité le 09/12/2022).

[Lee+20] Ryong LEE, Rae-young JANG, Minwoo PARK, Ga-ye JEON, Jae-kwang KIM et Sang-hwan LEE. « Making IoT Data Ready for Smart City Applications ». In : *2020 IEEE International Conference on Big Data and Smart Computing (BigComp)*. Fév. 2020, p. 605-608. DOI : 10.1109/BigComp48618.2020.00020.

[LLR13] Tayeb LEMLOUMA, Sébastien LABORIE et Philippe ROOSE. « Toward a Context-Aware and Automatic Evaluation of Elderly Dependency in Smart Homes and Cities ». In : *2013 IEEE 14th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*. IEEE, 2013, p. 1-6.

[LoR16] LORA ALLIANCE®. *LoRaWAN Security Whitepaper*. Juill. 2016. (Visité le 18/01/2021).

[LZ21] Karim LOUNIS et Mohammad ZULKERNINE. « T2T-MAP : A PUF-Based Thing-to-Thing Mutual Authentication Protocol for IoT ». In : *IEEE Access* 9 (2021), p. 137384-137405. ISSN : 2169-3536. DOI : 10.1109/ACCESS.2021.3117444.

[Mae12] Roel Maes. « Physically Unclonable Functions : Constructions, Properties and Applications (Fysisch onkloonbare functies : constructies, eigenschappen en toepassingen) ». Thèse de doct. 2012. (Visité le 13/08/2021).

[Mar+17] Cédric Marchand, Lilian Bossuet, Ugo Mureddu, Nathalie Bochard, Abdelkarim Cherkaoui et Viktor Fischer. « Implementation and Characterization of a Physical Unclonable Function for IoT : A Case Study With the TERO-PUF ». In : *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.1 (déc. 2017), p. 97-109. ISSN : 1937-4151. DOI : 10.1109/TCAD.2017.2702607.

[Mar99] W. Mark. « Turning Pervasive Computing into Mediated Spaces ». In : *IBM Systems Journal* 38.4 (1999), p. 677-692. ISSN : 0018-8670. DOI : 10.1147/sj.384.0677.

[Mau+12] Yoann Maurel, Stéphanie Chollet, Vincent Lestideau, Jonathan Bardin, Philippe Lalanda et André Bottaro. « fANFARE : Autonomic Framework for Service-Based Pervasive Environment ». In : *2012 IEEE Ninth International Conference on Services Computing*. Juin 2012, p. 65-72. DOI : 10.1109/SCC.2012.7.

[MES08] Giacomo V. Mc Evoy et Bruno Schulze. « Using Clouds to Address Grid Limitations ». In : *Proceedings of the 6th International Workshop on Middleware for Grid Computing*. Leuven Belgium : ACM, déc. 2008, p. 1-6. ISBN : 978-1-60558-365-5. DOI : 10.1145/1462704.1462715. (Visité le 26/01/2023).

[MMM21] Michael Chui, Mark Collins et Mark Patel. *Where and How to Capture Accelerating IoT Value*. https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/iot-value-set-to-accelerate-through-2030-where-and-how-to-capture-it. Nov. 2021. (Visité le 11/05/2023).

[Mil86] Victor S. Miller. « Use of Elliptic Curves in Cryptography ». In : *Advances in Cryptology — CRYPTO '85 Proceedings*. Sous la dir. d'Hugh C. Williams. T. 218. Berlin, Heidelberg : Springer Berlin Heidelberg, 1986, p. 417-426. ISBN : 978-3-540-16463-0. DOI : 10.1007/3-540-39799-X_31. (Visité le 12/12/2022).

[MPL21] Partemie-Marian Mutescu, Adrian Ioan Petrariu et Alexandru Lavric. « Wireless Communications for IoT : Energy Efficiency Survey ». In : *2021 12th International Symposium on Advanced Topics in Electrical Engineering (ATEE)*. Mars 2021, p. 1-4. DOI : 10.1109/ATEE52255.2021.9425330.

[Nir15] Yoav Nir. *ChaCha20, Poly1305, and Their Use in the Internet Key Exchange Protocol (IKE) and IPsec*. Request for Comments RFC 7634. Internet Engineering Task Force, août 2015. DOI : 10.17487/RFC7634. (Visité le 09/12/2022).

[NL15] Yoav Nir et Adam Langley. *ChaCha20 and Poly1305 for IETF Protocols*. Request for Comments RFC 7539. Internet Engineering Task Force, mai 2015. DOI : 10.17487/RFC7539. (Visité le 09/12/2022).

[OAS] OASIS. *Advanced Message Queuing Protocol (AMQP) Version 1.0*. http://docs.oasis-open.org/amqp/core/v1.0/amqp-core-complete-v1.0.pdf. (Visité le 08/07/2023).

[OAS04] OASIS. *Web Services Security : SOAP Message Security 1.1*. https://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf. Fév. 2004. (Visité le 01/04/2022).

[Omo+19] B. Omoniwa, R. Hussain, M. A. Javed, S. H. Bouk et S. A. Malik. « Fog/Edge Computing-Based IoT (FECIoT) : Architecture, Applications, and Research Issues ». In : *IEEE Internet of Things Journal* 6.3 (juin 2019), p. 4118-4149. ISSN : 2327-4662. DOI : 10.1109/JIOT.2018.2875544.

[PM18] Jianli Pan et James McElhannon. « Future Edge Cloud and Edge Computing for Internet of Things Applications ». In : *IEEE Internet of Things Journal* 5.1 (fév. 2018), p. 439-449. ISSN : 2327-4662. DOI : 10.1109/JIOT.2017.2767608.

[Pap+20] Athanasios Papadimitriou, Konstantinos Nomikos, Mihalis Psarakis, Ehsan Aerabi et David Hely. « You Can Detect but You Cannot Hide : Fault Assisted Side Channel Analysis on Protected Software-based Block Ciphers ». In : *2020 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. Oct. 2020, p. 1-6. DOI : 10.1109/DFT50435.2020.9250870.

[Pap03] M.P. Papazoglou. « Service-Oriented Computing : Concepts, Characteristics and Directions ». In : *Proceedings of the Fourth International Conference on Web Information Systems Engineering, 2003. WISE 2003.* Déc. 2003, p. 3-12. DOI : 10.1109/WISE.2003.1254461.

[Pap01] Ravikanth Srinivasa Pappu. « Physical One-Way Functions ». Thesis. Massachusetts Institute of Technology, 2001. (Visité le 11/06/2021).

[Pie+20] Paola Pierleoni, Roberto Concetti, Alberto Belli et Lorenzo Palma. « Amazon, Google and Microsoft Solutions for IoT : Architectures and a Performance Comparison ». In : *Ieee Access* 8 (2020), p. 5455-5470. ISSN : 2169-3536. DOI : 10.1109/ACCESS.2019.2961511.

[QM10] Li Qilin et Zhou Mintian. « The State of the Art in Middleware ». In : *2010 International Forum on Information Technology and Applications*. T. 1. Juill. 2010, p. 83-85. DOI : 10.1109/IFITA.2010.118.

[Raz+16] Mohammad Abdur Razzaque, Marija Milojevic-Jevric, Andrei Palade et Siobhán Clarke. « Middleware for Internet of Things : A Survey ». In : *IEEE Internet of Things Journal* 3.1 (fév. 2016), p. 70-95. ISSN : 2327-4662. DOI : 10.1109/JIOT.2015.2498900.

[Res18] Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. Request for Comments RFC 8446. Internet Engineering Task Force, août 2018. DOI : 10.17487/RFC8446. (Visité le 21/03/2023).

[RTM22] Eric Rescorla, Hannes Tschofenig et Nagena Modadugu. *The Datagram Transport Layer Security (DTLS) Protocol Version 1.3*. Request for Comments RFC 9147. Internet Engineering Task Force, avr. 2022. DOI : 10.17487/RFC9147. (Visité le 07/03/2023).

[RTB20] Gabriele Restuccia, Hannes Tschofenig et Emmanuel Baccelli. « Low-Power IoT Communication Security : On the Performance of DTLS and TLS 1.3 ». In : *2020 9th IFIP International Conference on Performance Evaluation and Modeling in Wireless Networks (PEMWN)*. Déc. 2020, p. 1-6. DOI : 10.23919/PEMWN50727.2020.9293085.

[RSA78] R. L. Rivest, A. Shamir et L. Adleman. « A Method for Obtaining Digital Signatures and Public-Key Cryptosystems ». In : *Communications of the ACM* 21.2 (fév. 1978), p. 120-126. ISSN : 0001-0782. DOI : 10.1145/359340.359342. (Visité le 07/07/2023).

[RS16] Eyal Ronen et Adi Shamir. « Extended Functionality Attacks on IoT Devices : The Case of Smart Lights ». In : *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. Mars 2016, p. 3-12. DOI : 10.1109/EuroSP.2016.13.

[Rot+18] Felix Maximilian Roth, Christian Becker, German Vega et Philippe Lalanda. « XWARE—A Customizable Interoperability Framework for Pervasive Computing Systems ». In : *Pervasive and Mobile Computing* 47 (juill. 2018), p. 13-30. ISSN : 1574-1192. DOI : 10.1016/j.pmcj.2018.03.005. (Visité le 25/04/2023).

[RA17] Sudhir K. Routray et Sharath Anand. « Narrowband IoT for Healthcare ». In : *2017 International Conference on Information Communication and Embedded Systems (ICICES)*. Fév. 2017, p. 1-4. DOI : 10.1109/ICICES.2017.8070747.

[Rud17] Vishruta Rudresh. *ZigBee Security : Basics (Part 2)*. Nov. 2017. (Visité le 19/02/2021).

[RH14] U. Rührmair et D. E. Holcomb. « PUFs at a Glance ». In : *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*. Mars 2014, p. 1-6. DOI : 10.7873/DATE.2014.360.

[Rüh+10] Ulrich Rührmair, Frank Sehnke, Jan Sölter, Gideon Dror, Srinivas Devadas et Jürgen Schmidhuber. « Modeling Attacks on Physical Unclonable Functions ». In : *Proceedings of the 17th ACM Conference on Computer and Communications Security*. CCS '10. New York, NY, USA : Association for Computing Machinery, oct. 2010, p. 237-249. ISBN : 978-1-4503-0245-6. DOI : 10.1145/1866307.1866335. (Visité le 07/12/2020).

[RS14] Ulrich Rührmair et Jan Sölter. « PUF Modeling Attacks : An Introduction and Overview ». In : *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*. Mars 2014, p. 1-6. DOI : 10.7873/DATE.2014.361.

[SM03] D. Saha et A. Mukherjee. « Pervasive Computing : A Paradigm for the 21st Century ». In : *Computer* 36.3 (mars 2003), p. 25-31. ISSN : 1558-0814. DOI : 10.1109/MC.2003.1185214.

[San+19] Aellison C. T. Santos, José L. Soares Filho, Ávilla Í. S. Silva, Vivek Nigam et Iguatemi E. Fonseca. « BLE Injection-Free Attack : A Novel Attack on Bluetooth Low Energy Devices ». In : *Journal of Ambient Intelligence and Humanized Computing* (sept. 2019). issn : 1868-5145. doi : 10.1007/s12652-019-01502-z. (Visité le 14/01/2021).

[Sar+22] Iqbal H. Sarker, Asif Irshad Khan, Yoosef B. Abushark et Fawaz Alsolami. « Internet of Things (IoT) Security Intelligence : A Comprehensive Overview, Machine Learning Solutions and Research Directions ». In : *Mobile Networks and Applications* (mars 2022). issn : 1572-8153. doi : 10.1007/s11036-022-01937-3. (Visité le 07/06/2023).

[Sat+09] Mahadev Satyanarayanan, Paramvir Bahl, Ramon Caceres et Nigel Davies. « The Case for VM-Based Cloudlets in Mobile Computing ». In : *IEEE Pervasive Computing* 8.4 (oct. 2009), p. 14-23. issn : 1558-2590. doi : 10.1109/MPRV.2009.82.

[Seb+16] A. Sebbar, S. Boulahya, G. Mezzour et M. Boulmalf. « An Empirical Study of WIFI Security and Performance in Morocco - Wardriving in Rabat ». In : *2016 International Conference on Electrical and Information Technologies (ICEIT)*. Mai 2016, p. 362-367. doi : 10.1109/EITech.2016.7519621.

[Shi+16] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li et Lanyu Xu. « Edge Computing : Vision and Challenges ». In : *IEEE Internet of Things Journal* 3.5 (oct. 2016), p. 637-646. issn : 2327-4662. doi : 10.1109/JIOT.2016.2579198.

[SGP18] Georg Sigl, Mathieu Gross et Michael Pehl. « Where Technology Meets Security : Key Storage and Data Separation for System-on-Chips ». In : *ESSCIRC 2018 - IEEE 44th European Solid State Circuits Conference (ESSCIRC)*. Sept. 2018, p. 12-17. doi : 10.1109/ESSCIRC.2018.8494319.

[SD07] G. E. Suh et S. Devadas. « Physical Unclonable Functions for Device Authentication and Secret Key Generation ». In : *2007 44th ACM/IEEE Design Automation Conference*. San Diego, California : Association for Computing Machinery, juin 2007, p. 9-14. isbn : 978-1-59593-627-1. doi : 10.1145/1278480.1278484.

[TS19] Varun M Tayur et R Suchithra. « A Comprehensive Ontology for Internet of Things (COIoT) ». In : *2019 Second International Conference on Advanced Computational and Communication Paradigms (ICACCP)*. Fév. 2019, p. 1-6. doi : 10.1109/ICACCP.2019.8882936.

[VH+12] Anthony Van Herrewege, Stefan Katzenbeisser, Roel Maes, Roel Peeters, Ahmad-Reza Sadeghi, Ingrid Verbauwhede et Christian Wachsmann. « Reverse Fuzzy Extractors : Enabling Lightweight Mutual Authentication for PUF-Enabled RFIDs ». In : *Financial Cryptography and Data Security*. Sous la dir. d'Angelos D. Keromytis. Lecture Notes in Computer Science. Berlin, Heidelberg : Springer, jan. 2012, p. 374-389. isbn : 978-3-642-32946-3. doi : 10.1007/978-3-642-32946-3_27.

[Vas14] J. P. VASSEUR. *Terms Used in Routing for Low-Power and Lossy Networks*. Request for Comments RFC 7102. Internet Engineering Task Force, jan. 2014. DOI : 10.17487/RFC7102. (Visité le 23/01/2023).

[W3C04] W3C. *Web Services Architecture*. https://www.w3.org/TR/ws-arch/. Fév. 2004. (Visité le 04/10/2021).

[W3C07a] W3C. *SOAP Version 1.2 Part 1 : Messaging Framework (Second Edition)*. https://www.w3.org/TR/soap12/. Avr. 2007. (Visité le 29/03/2022).

[W3C07b] W3C. *Web Services Description Language (WSDL) Version 2.0 Part 1 : Core Language*. https://www.w3.org/TR/wsdl/. Juin 2007. (Visité le 21/03/2022).

[W3C07c] W3C. *Web Services Policy 1.5 - Framework*. https://www.w3.org/TR/ws-policy/. Sept. 2007. (Visité le 21/03/2022).

[Wei91] Mark WEISER. « The Computer for the 21st Century ». In : *ACM SIGMOBILE Mobile Computing and Communications Review* 3.3 (juill. 1991), p. 3-11. ISSN : 1559-1662, 1931-1222. DOI : 10.1145/329124.329126. (Visité le 08/04/2021).

[Wri09] Joshua WRIGHT. *KillerBee : Practical ZigBee Exploitation Framework*. https://www.willhackforsushi.com/presentations/toorcon11-wright.pdf. 2009. (Visité le 02/12/2020).

[Wu+20] Jianliang WU, Yuhong NAN, Vireshwar KUMAR, Dave (Jing) TIAN, Antonio BIANCHI, Mathias PAYER et Dongyan XU. « BLESA : Spoofing Attacks against Reconnections in Bluetooth Low Energy ». In : *14th {USENIX} Workshop on Offensive Technologies ({WOOT} 20)*. 2020. (Visité le 06/01/2021).

[Yaq+18] LV YAQIONG, TU LEI, C.K.M LEE et TANG XIN. « IoT Based Omni-Channel Logistics Service in Industry 4.0 ». In : *2018 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI)*. Juill. 2018, p. 240-243. DOI : 10.1109/SOLI.2018.8476708.

[Zet] Kim ZETTER. « A Cyberattack Has Caused Confirmed Physical Damage for the Second Time Ever ». In : *Wired* (). ISSN : 1059-1028. (Visité le 06/02/2023).

[Zha+17] Nan ZHANG et al. « Understanding IoT Security Through the Data Crystal Ball : Where We Are Now and Where We Are Going to Be ». In : *arXiv :1703.09809 [cs]* (mars 2017). arXiv : 1703.09809 [cs]. (Visité le 04/05/2021).

[Zig15] ZIGBEE ALLIANCE. *ZigBee Specification*. https://zigbeealliance.org/wp-content/uploads/2019/11/docs-05-3474-21-0csg-zigbee-specification.pdf. Août 2015. (Visité le 07/11/2022).

[ZS15] Tobias ZILLNER et Sebastian STROBL. *ZigBee Exploited - The Good, the Bad, the Ugly*. https://www.blackhat.com/docs/us-15/materials/us-15-Zillner-ZigBee-Exploited-The-Good-The-Bad-And-The-Ugly.pdf. 2015. (Visité le 02/12/2020).

[da +21] Mauro A. A. DA CRUZ, Joel J. P. C. RODRIGUES, Pascal LORENZ, Valery V. KOROTAEV et Victor Hugo C. DE ALBUQUERQUE. « In.IoT—A New Middleware for Internet of Things ». In : *IEEE Internet of Things Journal* 8.10 (mai 2021), p. 7902-7911. ISSN : 2327-4662. DOI : 10.1109/JIOT.2020.3041699.

[neh] NEHSIN. *Présentation des jumeaux d'appareil Azure IoT Hub*. https://docs.microsoft.com/fr-fr/azure/iot-hub/iot-hub-devguide-device-twins. (Visité le 22/02/2021).

[phi] PHILMEA. *Présentation des méthodes directes Azure IoT Hub*. https://docs.microsoft.com/fr-fr/azure/iot-hub/iot-hub-devguide-direct-methods. (Visité le 30/03/2021).

[wes] WESMC7777. *Vue d'ensemble du service Azure IoT Hub Device Provisioning*. https://docs.microsoft.com/fr-fr/azure/iot-dps/about-iot-dps. (Visité le 26/03/2021).