République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université A/Mira de Béjaïa

Faculté des Sciences Exactes
Département d'Informatique

Mémoire de fin d'études

En vue de l'obtention du diplôme de Master en Informatique
Option : Réseaux et Sécurité
Administration et Sécurité des Réseaux

Thème :

Sécurisation des environnements IoT à base de WSN

Réalisé par :
M^{lle}Ben Mokhtar Ghenima
M^{lle} Dellys Amel

Encadré par :
Zoubeyr Farah Encadrant - Professeur - Université de Béjaïa.

Soutenu devant le jury composé de :

GOUDJIL Slimane	Président	Université de Béjaïa.
MIR Foudil	Examinateur	Université de Béjaïa.
SLIMANI Hachem	Examinateur	Université de Béjaïa.
BOUZIDI Zair	Examinateur	Université de Béjaïa.

Promotion 2024 - 2025

Résumé

Ce mémoire s'inscrit dans le cadre des recherches actuelles sur la sécurisation des réseaux IoT à faibles ressources, un domaine crucial face à la croissance continue des dispositifs connectés dans notre quotidien. L'Internet des Objets (IoT) permet à des appareils physiques de détecter, collecter et transmettre des informations via Internet. Cependant, l'application des protocoles de sécurité standards reste difficile en raison des contraintes matérielles propres à ces appareils.

Initialement, cette étude s'est consacrée à l'exposé des principes de base de l'IoT, en détaillant son architecture, ses restrictions matérielles et les dangers qui le guettent, particulièrement en ce qui concerne la gestion des clés cryptographiques. Par la suite, une étude approfondie du protocole EVKMS (Enhanced Vector-based Key Management Scheme) a été menée, mettant en évidence son intérêt pour les environnements contraints grâce à sa légèreté et son efficacité énergétique.

Dans le prolongement de cette recherche, nous avons proposé une amélioration du protocole, baptisée A-EVKMS. Cette version intègre un système de surveillance intelligente des nœuds ainsi qu'une modification du schéma de dérivation de clés. La solution a été validée par une implémentation sur un réseau de quinze nœuds, incluant l'implémentation d'attaques types et une évaluation des performances.

Les résultats obtenus démontrent l'efficacité de notre solution en termes de sécurité, d'occupation mémoire et de temps de réponse. Ce travail constitue ainsi une contribution vers des mécanismes de sécurité légers, évolutifs et performants pour les objets connectés à ressources limitées, tout en ouvrant la voie à de futures expérimentations sur des dispositifs physiques et des protocoles de communication standardisés.

Mots-clés : IoT, sécurité, gestion des clés, EVKMS, objets connectés, protocole léger

Abstract

This Master thesis is part of current research into securing low-resource IoT networks, a critical area in the face of the proliferation of connected objects in our daily lives. The Internet of Things (IoT) enables physical devices to sense, collect, and transmit data over the Internet. However, the limited resources of these devices complicate the implementation of traditional security protocols.

This work begins by presenting the fundamentals of IoT, detailing its architecture, hardware constraints, and the various threats it faces—particularly in key management. Then, we conducted an in-depth analysis of the EVKMS (Enhanced Vector-based Key Management Scheme) protocol, which is well suited for constrained environments due to its lightweight nature and energy efficiency.

Building upon this foundation, we proposed an enhanced version named A-EVKMS. This version integrates a sleep scheduling mechanism and a modified key derivation scheme. The proposed solution was validated through an implementation on a fifteen-node network, including typical attack scenarios and a performance evaluation.

The results confirm the solution's effectiveness in terms of security, memory usage, and response time. This work therefore contributes to the development of lightweight, scalable, and efficient security solutions for constrained IoT devices, paving the way for future experiments on real hardware and integration with standardized communication protocols.

Keywords : IoT, security, key management, EVKMS, constrained devices, lightweight protocol

Remerciements

En tout premier lieu, nous souhaitons adresser nos sincères remerciements à Monsieur Zoubeyr Farah, notre encadrant, pour sa disponibilité, ses conseils précieux et son accompagnement tout au long de ce mémoire. Sa rigueur, sa bienveillance et son engagement ont grandement contribué à la réussite de ce travail, et nous lui en sommes profondément reconnaissantes.
Nous remercions également Monsieur GOUDJIL Slimane, Président du jury, ainsi que Messieurs MIR Foudil, SLIMANI Hachem et BOUZIDI Zair, membres du jury, pour l'intérêt qu'ils ont porté à notre projet et pour leur évaluation attentive et bienveillante.
Nos remerciements vont aussi à nos parents, pour leur soutien indéfectible, leur patience, et leur confiance. Leur présence à nos côtés, tout au long de ce parcours, a été un véritable pilier.Nous pensons également à nos familles, proches et amies, pour leurs encouragements, leur écoute et leur bonne humeur, qui nous ont porté tout au long de cette aventure.
Enfin, nous remercions toutes les personnes qui, de près ou de loin, ont contribué à la réalisation de ce mémoire.

Dédicaces

Nous dédions ce mémoire à tous ceux qui nous entourent avec bienveillance et amour. À nos familles, pour leur soutien indéfectible, leur patience et leurs encouragements constants, qui ont été notre force tout au long de ce parcours.

Un hommage particulier à nos parents, véritables piliers de notre vie, dont la générosité et la confiance nous ont permis d'aller au bout de ce projet.

Notre reconnaissance s'étend également à nos cousins, cousines, oncles et tantes, qui, par leurs attentions et leurs paroles rassurantes, ont su nous soutenir dans les moments les plus exigeants.

Nous pensons aussi à nos amies, véritables sœurs de cœur, pour leur écoute, leur humour et leur présence réconfortante, qui ont su égayer les journées les plus longues.

Enfin, à toutes les personnes qui, d'une manière ou d'une autre, ont contribué à cette belle aventure humaine et académique, nous exprimons notre profonde gratitude.

Avec tendresse, respect et affection.

Amel & Ghenima

Table des matières

Table des figures

Liste des tableaux

Liste des abréviations

6LoWPAN : IPv6 over Low-Power Wireless Personal Area Networks
A-EVKMS : Adapted-Enhanced Vector-based Key Management Scheme
AES : Advanced Encryption Standard
ASCON : Algorithme de chiffrement léger sélectionné par le NIST
BLE : Bluetooth Low Energy
BS : Base Station (Station de base)
CH : Cluster Head (Chef de cluster)
CF : Cipher Feedback (mode de chiffrement par flot)
CM : Cluster Member (Membre de cluster)
CoAP : Constrained Application Protocol
DoS/DDoS : Denial of Service / Distributed Denial of Service
ECC : Elliptic Curve Cryptography
EVKMS : Enhanced Vector-based Key Management Scheme
GUID : Globally Unique Identifier
HTTP : HyperText Transfer Protocol
ID : Identifiant
IEEE : Institute of Electrical and Electronics Engineers
IETF : Internet Engineering Task Force
IoT : Internet of Things (Internet des Objets)
IRTF : Internet Research Task Force
ISA : Intrusion Sensing Algorithm
ISO/IEC : International Organization for Standardization / International Electrotechnical Commission
KB : Kilobyte (kilooctet)
KDC : Key Distribution Center
KM : Key Manager (Gestionnaire de clés)
MAC : Message Authentication Code
MITM : Man-In-The-Middle (attaque de l'homme du milieu)
MQTT : Message Queuing Telemetry Transport
ms : Millisecondes
NIST : National Institute of Standards and Technology

PKG : Private Key Generator

QKD : Quantum Key Distribution

RFC : Request For Comments

RSA : Rivest–Shamir–Adleman

SHA-256 : Secure Hash Algorithm 256 bits

SKWN : Secure Key Management Wireless Network

TLS : Transport Layer Security

UIT : Union Internationale des Télécommunications

WSN : Wireless Sensor Network (réseau de capteurs sans fil)

ZKP : Zero-Knowledge Proof (preuve à divulgation nulle de connaissance)

Introduction Générale

Un monde intelligent, ou « planète intelligente », désigne un environnement où des objets connectés à Internet interagissent intelligemment entre eux et avec les humains. Cette interconnexion d'objets capables de percevoir, traiter et échanger des données en temps réel constitue ce qu'on appelle l'Internet des Objets (IoT). Il a redéfini notre rapport à la technologie, avec des applications variées telles que les maisons intelligentes, l'industrie automatisée ou encore le domaine médical. L'IoT s'est imposé dans notre quotidien, permettant à des objets physiques de collecter, traiter et transmettre des données via des réseaux, apportant confort et efficacité.

Cependant, cette interconnexion massive soulève des enjeux de sécurité majeurs, notamment dans les environnements à ressources limitées. Les objets connectés disposent souvent de capacités restreintes en termes de calcul, de mémoire et d'énergie, ce qui complique l'implémentation de mécanismes de sécurité robustes. Ces contraintes augmentent la surface d'attaque et rendent la protection contre les menaces plus complexe, faisant de la sécurité un enjeu central dans la conception des systèmes IoT.

Parmi les défis essentiels figure la gestion des clés cryptographiques, indispensable pour garantir la confidentialité, l'intégrité et l'authenticité des communications. Ce mémoire explore ce défi en se concentrant sur des solutions légères et adaptées aux contraintes des réseaux de capteurs. Pour ce faire, une analyse des architectures existantes a été menée, suivie de l'implémentation d'un mécanisme amélioré basé sur le protocole EVKMS, appelé AEVKMS.

Dans notre amélioration, nous avons intégré une stratégie de mise en veille intelligente (Sleep Scheduling) visant à optimiser la consommation énergétique des nœuds tout en maintenant un niveau de sécurité élevé. De plus, nous avons allégé le mécanisme dérivation des clés partagées pour le rendre plus adapté aux dispositifs contraints. Ces ajustements permettent à A-EVKMS d'assurer la sécurité des échanges tout en respectant les limitations des objets connectés.

Ce mémoire est structuré comme suit :

- **Chapitre 1 : Concepts fondamentaux sur l'IoT :** Il présente l'architecture typique des réseaux IoT, leurs caractéristiques ainsi que les principaux défis de sécurité associés.
- **Chapitre 2 : Implémentation du protocole EVKMS (Enhanced Vector-based Key :** Management Scheme) : Il analyse les différents schémas de gestion de clés existants dans la littérature, en mettant l'accent sur le protocole EVKMS, ses avantages et ses limites.
- **Chapitre 3 : Amélioration du protocole EVKMS (A-EVKMS) :** Ce chapitre constitue le cœur du travail. Il décrit notre amélioration du protocole EVKMS, expose ses motivations, présente l'architecture proposée, les étapes de mise en œuvre, et compare les performances du protocole amélioré avec la version d'origine.

Chapitre 1

Concepts fondamentaux sur l'IoT

1.1 Introduction

Depuis quelques années, nous assistons à l'essor fulgurant de l'Internet des Objets (IoT). Ce nouveau paradigme représente une révolution de l'Internet qui peut connecter les objets de notre quotidien à Internet et entre eux. Il est considéré comme une innovation technologique majeure dans l'industrie des nouvelles technologies de l'information et de la communication. L'IoT n'a pas une définition unique mais d'une manière générale, il est défini comme étant une extension de l'Internet actuel à tous les objets pouvant communiquer de manière directe ou indirecte avec des équipements électroniques, eux-mêmes connectés à Internet [5]. Ce chapitre constitue un élément clé de notre réflexion, fournissant une vue d'ensemble approfondie des principes fondamentaux de l'IoT. Il débute par une analyse rigoureuse des différentes définitions de l'IoT, mettant en évidence les multiples perspectives qui façonnent ce concept. Ces définitions établissent ainsi le socle de notre compréhension de l'IoT.

1.2 Réseaux de capteurs IoT

L'Internet des objets (IoT) consiste à connecter des objets physiques, tels que des capteurs et des appareils, à Internet. Ces appareils sont dotés de capacités de communication leur permettant d'échanger des données entre eux et avec leur environnement via le réseau, sans intervention humaine. Les réseaux IoT sont dynamiques, les appareils les rejoignent et les quittent fréquemment. Les applications IoT sont de plus en plus utilisées dans l'industrie, la santé, l'agriculture, les transports et la gestion de l'énergie.[1].

Le concept d'Internet des objets a une histoire bien établie, avec diverses interprétations et définitions. Comme l'a formulé l'Union internationale des télécommunications (UIT) en 2012 [10], l'IoT est caractérisé comme une infrastructure mondiale facilitant l'avancement de la société de l'information. Il y parvient en interconnectant des entités physiques et virtuelles grâce à des informations et des communications évolutives et interopérables.technologies de l'information. L'IoT incarne la vision d'une connectivité omniprésente, permettant l'interaction des personnes et des objets à tout moment, en tout lieu, par n'importe quel moyen et avec n'importe quel service, comme le résument idéalement Perera et al.

L'Internet des Objets (IoT) est défini par diverses organisations internationales, notamment l'ISO/IEC, l'IETF, l'IRTF et l'IEEE, comme un réseau d'objets physiques capables de collecter et de transmettre des données. Ces objets interconnectés, accompagnés d'humains et de ressources informationnelles, forment une infrastructure permettant le traitement des informations

et l'adaptation des actions en conséquence.

Les exigences de l'ISO/IEC pour l'IoT incluent l'auto-configuration, l'identification unique, la mobilité, la connectivité, la fiabilité et la standardisation des interfaces. L'IETF met en avant l'utilisation de protocoles standardisés pour garantir la communication entre les objets. De son côté, le groupe de recherche T2TRG de l'IRTF définit l'IoT comme un réseau où des nœuds à faibles ressources interagissent avec Internet pour favoriser l'innovation.

L'IEEE, à travers le projet P2413, ainsi que des acteurs industriels tels que Cisco, IBM, Huawei et Intel, contribuent à structurer l'IoT en tant que système interconnecté, adapté aux exigences spécifiques des différents secteurs. Le NIST considère l'IoT comme un système cyber-physique améliorant la qualité de vie via des objets intelligents, tandis que l'OASIS insiste sur l'adaptation des technologies existantes pour une meilleure intégration des capteurs et des réseaux physiques.

En résumé, l'IoT repose sur la connectivité des objets à Internet afin de proposer des services accessibles partout et à tout moment, en s'appuyant sur des standards et protocoles rigoureux pour garantir son efficacité et son évolution [6].

1.3 Architecture IoT

L'objectif fondamental de l'IoT étant la connexion d'un nombre considérable d'objets à Internet, il est essentiel de disposer d'une architecture bien définie pour assurer son bon fonctionnement. À ce jour, aucune architecture universelle ne fait consensus au sein de la communauté scientifique, bien que plusieurs modèles aient été proposés. Dans le cadre de ce mémoire, nous avons opté pour une architecture à trois couches : la couche de perception, la couche réseau et la couche application, dont nous allons détailler les caractéristiques dans les sections suivantes.[2]

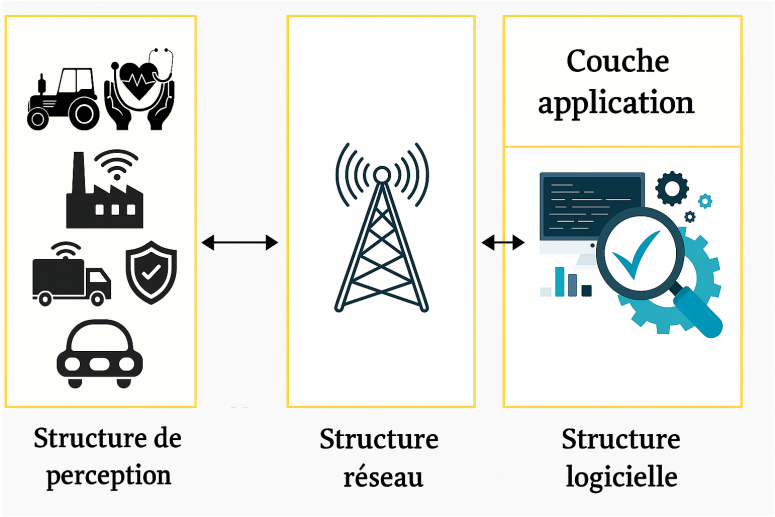


FIGURE 1.1 – Schéma général d'un réseau IoT

1.3.1 Couche Perception

La couche de perception est la couche la plus basse de l'architecture IoT. Elle est responsable de la collecte des données du monde réel. Les capteurs et actionneurs sans fil sont les principaux composants de cette couche.

Les informations ainsi collectées sont transmises à la couche réseau via des canaux sécurisés. La grande quantité de données générées par l'IoT provient de cette couche. Le coût des objets connectés et leur autonomie sont les contraintes de cette couche, en dehors des données massives qu'ils génèrent [4].

1.3.2 Couche Réseau

Occupant une position intermédiaire dans l'architecture de l'IoT, la couche réseau joue un rôle essentiel en assurant la transmission des données entre la couche de perception et la couche application. Parmi les technologies les plus couramment utilisées à ce niveau figurent Bluetooth Low Energy (BLE), Zigbee, WiFi, LoRaWAN (Long Range Wide Area Networks) et IPv6, notamment via les réseaux personnels sans fil à faible consommation (6LoWPAN).

- **WIFI :** Le Wi-Fi est une technologie de réseau sans fil qui permet aux périphériques tels que des ordinateurs (portables et fixes), des périphériques mobiles (téléphones intelligents et dispositifs portables), et d'autres équipements (imprimantes et caméras vidéo) d'accéder à Internet. Il permet à ces appareils, et à de nombreux autres, d'échanger des renseignements entre eux, ce qui crée un réseau qui peut prendre en charge de nombreux appareils simultanément. Il est notamment utilisé dans les maisons intelligentes, les villes intelligentes et dans le domaine de l'industrie [2].
- **Zigbee :** un protocole de communication sans fil basé sur la norme IEEE 802.15.4, conçu pour les applications à faible consommation d'énergie et à faible débit de données. Il est largement utilisé dans l'IoT, notamment en domotique, contrôle industriel et soins de santé, grâce à son efficacité énergétique, sa fiabilité et sa capacité d'évolution des réseaux. L'une de ses principales caractéristiques est sa topologie en maillage, où chaque appareil peut relayer des données vers d'autres, améliorant ainsi la portée et la robustesse du réseau. En cas d'indisponibilité d'un appareil, les données peuvent toujours être acheminées via d'autres nœuds du maillage. Cependant, Zigbee n'est pas adapté aux applications nécessitant une bande passante élevée, comme le streaming vidéo.Son interopérabilité est assurée par la Zigbee Alliance, une organisation qui définit les normes garantissant que les appareils Zigbee de différents fabricants peuvent communiquer entre eux.
- **Bluetooth :** désigne une norme de communication sans fil par ondes radio capable de transmettre des données et de la voix entre deux appareils électroniques compatibles. Le Bluetooth est notamment très répandu sur les téléphones mobiles, les écouteurs et casques sans fil ou encore les enceintes nomades. Il fonctionne sur les bandes de fréquence 2,4 GHz avec une portée maximale qui varie de 10 à 100 mètres [10].
- **BLE :** Il s'agit d'une technologie de réseau personnel sans fil utilisée pour les communications à courte portée. Cette technologie à faible consommation d'énergie trouve de nombreuses applications dans de nombreux contextes IoT, notamment, mais sans s'y limiter, les environnements résidentiels intelligents, les infrastructures urbaines intelligentes et les systèmes de santé. La principale différence entre BLE et Bluetooth est que le BLE est conçu pour une faible consommation d'énergie [7].
- **LoRaWAN :** est un protocole de communication sans fil conçu pour les applications IoT et machine-to-machine (M2M) nécessitant une connectivité longue portée et une faible consommation d'énergie. Basé sur la technologie LoRa, il est particulièrement adapté aux scénarios où de faibles volumes de données doivent être transmis sur de grandes distances tout en minimisant la dépense énergétique. Parmi ses principales caractéristiques, LoRaWAN offre une communication bidirectionnelle, permettant aux appareils

non seulement d'envoyer des données, mais aussi d'en recevoir, ce qui améliore leur fonctionnalité et leur capacité d'adaptation aux conditions du terrain. Son évolutivité est un autre atout majeur, puisqu'il peut intégrer un grand nombre d'appareils sans compromettre les performances du réseau. LoRaWAN adopte une topologie en étoile, où chaque appareil est connecté à une passerelle, elle-même reliée à un serveur réseau. Cette architecture facilite la gestion des communications et assure une transmission efficace des données collectées par les capteurs. Grâce à ces caractéristiques, LoRaWAN constitue une solution idéale pour les applications nécessitant une connectivité fiable sur de vastes zones géographiques [1].

- **6LoWPAN :** Il s'agit d'un protocole de communication sans fil basé sur le protocole IPv6 [33]. Il permet l'envoi et la réception de paquets IPv6 sur des réseaux sans fil à faible consommation d'énergie et est spécialement conçu pour une utilisation avec des réseaux à faible consommation d'énergie. appareils et réseaux, qui ont une puissance et mémoire et ressources de traitement limitées.

1.3.3 Couche Application

La couche Application représente l'aspect le plus visible de l'architecture de l'Internet des Objets. Elle regroupe l'ensemble des applications qui exploitent les données collectées par la couche Perception. Son rôle essentiel consiste à agréger, analyser et interpréter ces données afin d'offrir une variété de services adaptés aux besoins des utilisateurs.

La couche d'application IoT sert d'interface entre l'utilisateur final et l'IoT,constitue le centre névralgique des systèmes IoT. Elle orchestre plusieurs fonctions critiques pour transformer les données brutes en services intelligents et exploitables. Elle assure notamment :

- **Traitement et analyse des données :** Transformation des données collectées (via capteurs/actionneurs) en informations exploitables.
- **Logique métier et prise de décision :** Mise en œuvre de règles, d'algorithmes et de logique pour automatiser les actions à partir des données traitées.
- **Déploiement des services d'application :** Développement d'applications IoT spécifiques aux domaines comme la domotique, l'industrie, l'agriculture, ou l'environnement.
- **Stockage et gestion des données :** Conservation des données historiques dans des bases dédiées pour l'analyse, les prévisions et la prise de décision à long terme.
- **Interfaces utilisateur :** Intégration d'outils comme des tableaux de bord web, apps mobiles ou logicielles pour l'interaction avec les utilisateurs.
- **Intégration avec des systèmes externes :** Connexion aux services cloud, API tierces ou systèmes d'entreprise pour un échange fluide des données.
- **Évolutivité :** Conception adaptée à la croissance des déploiements et à l'augmentation du volume de données et d'utilisateurs.
- **Sécurité :** Implémentation de mécanismes d'authentification, de chiffrement et de contrôle d'accès pour protéger les données et les dispositifs.
- **Traitement en temps réel :** Réactivité instantanée aux événements critiques grâce à des capacités d'analyse immédiate.
- **Protocoles de communication :** Utilisation de protocoles tels que MQTT, CoAP ou HTTP pour garantir un transfert de données efficace.

1.4 Caractéristiques des nœuds de capteurs IoT

Les nœuds de capteurs qui constituent la base des réseaux IoT sont des composants électroniques miniaturisés conçus pour être intégrés dans des objets physiques dans le but de collecter des informations spécifiques et de les communiquer entre eux et/ou avec la station de base. Pour assurer ces tâches, chaque nœud de capteurs est doté de Un nœud de capteurs dans un réseau IoT est généralement composé des éléments suivants [1] :

- **Unité d'acquisition :** elle mesure des grandeurs physiques comme l'humidité, la température ou la pression.
- **Unité de traitement :** elle collecte, vérifie et stocke les données, puis les prépare pour la transmission. Cette unité dispose de capacités limitées en termes de calcul, de mémoire et de traitement, et ces ressources ne sont généralement pas extensibles.
- **Unité de communication :** constituée d'un émetteur et d'un récepteur, elle permet l'échange de données avec la station de base et les autres nœuds via des liaisons radio, filaires ou non filaires. Sa portée étant restreinte, une antenne externe peut être nécessaire pour améliorer la transmission.
- **Source d'énergie :** une batterie alimente l'ensemble des composants. La durée de fonctionnement du nœud dépend directement de sa capacité énergétique.
- **Système de localisation :** basé sur le GPS ou un algorithme interne, il permet de situer géographiquement le capteur.
- **Unité de mobilité :** permet au nœud de se déplacer pour exécuter ses tâches, elle dispose de sa propre alimentation indépendante de la batterie principale.

En conséquent, les nœuds de capteurs sont caractérisés par [1] :

- Une capacité de stockage et de traitement très restreinte.
- Une courte portée de transmission liée aux limitations de la puissance du signal et de la capacité de rayonnement des antennes utilisées.
- Une énergie limitée : En général, les batteries utilisées ont une capacité énergétique très limitée. De plus, dans la plupart des applications, il est problématique, voire impossible, de recharger ou de changer la batterie. Et lorsque cette dernière est épuisée, le nœud de capteurs ne peut fournir aucun service et il est considéré mort et il est instantanément retiré du réseau.

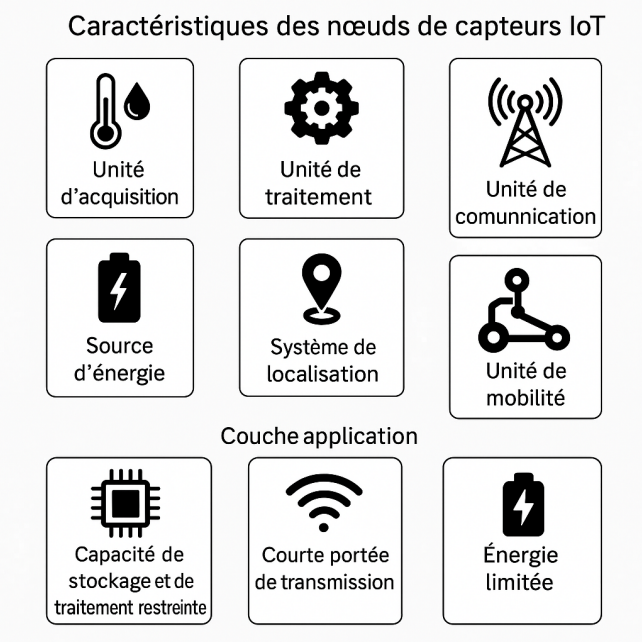
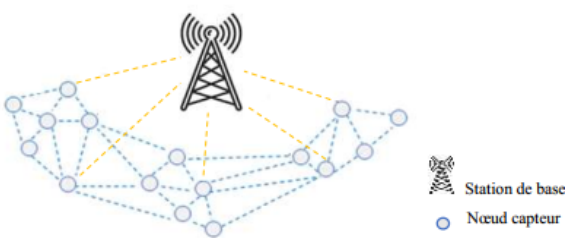


FIGURE 1.2 – Caractéristiques des nœuds de capteurs IoT

1.5 Spécificités des capteurs

Dans les réseaux IoT, les capteurs occupent une place essentielle. Ce sont des dispositifs chargés de mesurer des paramètres physiques tels que la température, l'humidité ou le mouvement, puis de transmettre ces données vers d'autres éléments du réseau, comme les passerelles ou les serveurs. Conçus pour être compacts, économiques et facilement déployables à grande échelle, ces capteurs sont toutefois fortement limités en termes de puissance de calcul, de mémoire et d'autonomie énergétique.

Dans une configuration simple, tous les nœuds capteurs sont placés à proximité de la station de base, ce qui permet un envoi direct des données en un seul saut. En revanche, dans les réseaux plus étendus couvrant de larges zones géographiques, certains capteurs se trouvent trop éloignés pour établir une communication directe. Dans ce cas, les données doivent être relayées à travers d'autres nœuds intermédiaires. Ainsi, deux types de topologies peuvent être utilisées pour garantir la transmission efficace des données vers la station de base [8].



Topologie plate

Topologie hiérarchique

FIGURE 1.3 – Images Topologie plate et hiérarchique

Les capteurs IoT sont, par nature, des dispositifs à ressources limitées. Leurs capacités de

calcul, de mémoire et de traitement sont restreintes, car leur conception privilégie la compacité, l'économie d'énergie et les faibles coûts de production. Cette architecture minimaliste les rend inadaptés à l'exécution de protocoles cryptographiques complexes, ce qui complique considérablement la sécurisation des communications. Les méthodes de chiffrement traditionnelles, trop lourdes pour leur architecture, constituent ainsi un obstacle majeur à surmonter.

À cela s'ajoute une contrainte énergétique critique. Généralement alimentés par des batteries non rechargeables ou des sources d'énergie réduites, ces capteurs doivent fonctionner sur de longues durées, souvent dans des zones difficilement accessibles. Or, toute opération gourmande en énergie telle que le chiffrement, la signature numérique ou l'émission fréquente de données peut accélérer la décharge des batteries, nuire à la stabilité du réseau et provoquer des pannes prématurées.

Ces limitations imposent donc le recours à des protocoles de sécurité allégés, optimisés pour consommer le moins possible de ressources matérielles. Il devient également indispensable de mettre en place des stratégies d'économie d'énergie, comme la mise en veille intelligente ou la limitation des communications superflues.

En définitive, garantir la sécurité dans un environnement IoT exige une approche adaptée aux réalités matérielles de ces dispositifs. Cela implique de trouver un équilibre judicieux entre le niveau de sécurité recherché et les capacités effectives des capteurs à le soutenir.

1.6 Défis de sécurité de l'IoT

Les appareils IoT ont profondément transformé le paysage numérique moderne, en rendant possibles une large gamme d'applications, allant des maisons intelligentes à l'automatisation industrielle. Toutefois, cette prolifération rapide s'accompagne d'un défi majeur : celui des ressources limitées en calcul, mémoire et énergie, qui caractérisent ces dispositifs. Ces contraintes représentent un enjeu central, notamment en matière de sécurité, où la garantie de la confidentialité et de l'intégrité des données dépend fortement des capacités matérielles des appareils. Les dispositifs IoT sont conçus pour être efficaces, compacts et peu énergivores, mais cette conception les rend vulnérables à diverses menaces, notamment à cause de leur incapacité à exécuter des algorithmes cryptographiques complexes.

De plus, leur mémoire limitée complique la gestion et le stockage sécurisé des clés, accentuant les risques d'accès non autorisé, surtout dans des environnements distants ou non surveillés. L'approvisionnement énergétique restreint, souvent basé sur des batteries à longue durée de vie, ajoute une contrainte supplémentaire, car les mécanismes de sécurité énergivores peuvent compromettre leur autonomie. Ainsi, dans le contexte de la cybersécurité IoT, prendre en compte ces limitations matérielles n'est pas un simple choix d'optimisation, mais une exigence fondamentale.

Ignorer ces contraintes peut ouvrir la voie à des failles exploitables par des acteurs malveillants. C'est pourquoi tout protocole de sécurité destiné à l'IoT doit être pensé pour concilier robustesse, légèreté et efficacité énergétique. Les sections suivantes de cette revue se pencheront sur les approches proposées par les systèmes de gestion de clés existants, en évaluant leur capacité à optimiser l'usage des ressources et à renforcer la durabilité et la sécurité des écosystèmes IoT.

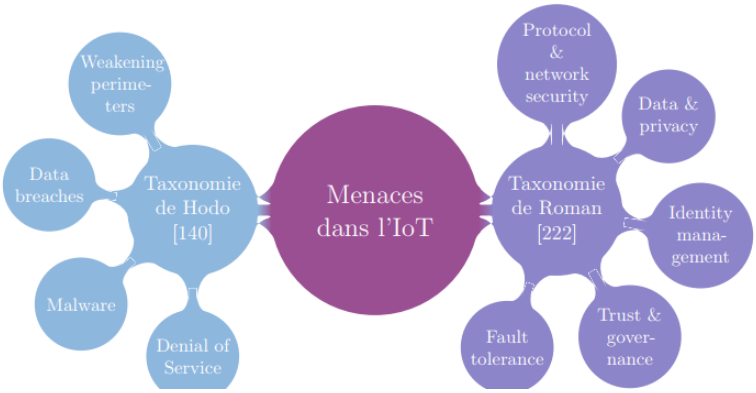


FIGURE 1.4 – Menaces dans L'IoT

1.6.1 Types d'attaques

La multitude de menaces qui pèsent sur l'Internet des Objets et les vulnérabilités qui s'ensuivent rendent possibles de nombreuses attaques sur ces systèmes. Les éléments de l'IoT sont sensibles aux mêmes attaques que les systèmes informatiques, en raison de leurs contraintes matérielles, de leur hétérogénéité et de leur dépendance aux communications sans fil.

1. **Attaque par Capture de Nœuds :** Un attaquant s'empare physiquement d'un appareil IoT (capteur, passerelle) pour voler ses données internes (clés cryptographiques, identifiants). Cette attaque est particulièrement dangereuse dans les systèmes utilisant des clés pré-partagées (comme le schéma de Blom), où la compromission d'un seul nœud peut exposer les clés de communication de tout un groupe. Les contre-mesures incluent l'effacement sécurisé des clés après leur utilisation et la protection matérielle (TPM).
2. **Attaque par Écoute Clandestine (Sniffing) :** L'attaquant intercepte les communications sans fil non chiffrées (Wi-Fi, Zigbee, LoRaWAN) pour voler des données sensibles. Par exemple, un capteur envoyant des mesures en clair via MQTT peut être espionné. Les solutions incluent le chiffrement obligatoire (AES, TLS) et l'authentification des messages (MAC).
3. **Attaque par Rejeu (Replay Attack) :** L'attaquant enregistre un message légitime (ex : "Ouvrir la porte") et le renvoie plus tard pour déclencher une action non autorisée. Pour l'éviter, les systèmes IoT doivent utiliser des nonces (nombres aléatoires uniques) ou des horodatages pour rendre chaque message unique.
4. **Attaque par Déni de Service (DoS/DDoS) :** Un appareil IoT est submergé de requêtes frauduleuses jusqu'à épuisement de ses ressources (CPU, batterie). Par exemple, le botnet Mirai a exploité des caméras IoT pour lancer des attaques DDoS massives. Les défenses incluent la limitation du débit (rate limiting) et la détection d'anomalies.
5. **Attaque par Force Brute :** L'attaquant teste systématiquement des combinaisons de mots de passe ou de clés faibles. Pour s'en protéger, il faut utiliser des clés longues (AES-256) et bloquer les tentatives répétées.
6. **Attaque par Usurpation (Spoofing) :** Un appareil malveillant se fait passer pour un périphérique légitime. Cela peut conduire à des injections de fausses données ou à du détournement de trafic. L'authentification forte (certificats numériques) est essentielle pour l'éviter.

7. **Attaque par Factorisation (Cryptanalyse) :** L'attaquant exploite des vulnérabilités mathématiques dans les algorithmes cryptographiques. Les solutions modernes incluent des algorithmes résistants aux quantiques.

1.6.2 Contraintes liées aux ressources

- Limitation de la capacité de traitement : Les dispositifs IoT sont équipés de processeurs de faible puissance, qui ne peuvent pas exécuter des algorithmes cryptographiques sophistiqués (par exemple : RSA, ECC). Cela restreint l'emploi de techniques de cryptage solides.
- Capacité restreinte en stockant des clés de chiffrement et des configurations de sécurité est limitée, ce qui complique la gestion des clés traditionnelles symétriques/asymétriques.
- Des attaques telles que l'écoute clandestine (sniffing) ou les attaques par replay peuvent cibler les communications sans fil, comme le Wi-Fi, le Bluetooth ou LoRaWAN.
- Authentification laxiste qui prédéfinis des procédures d'authentification élémentaires rendent la falsification d'identité (spoofing) plus aisée.

1.6.3 Problèmes de Gestion des Clés

Le secteur de la sécurité de l'IoT se distingue par son caractère dynamique et en perpétuel développement, alimenté par la présence grandissante des dispositifs connectés. Un élément crucial pour sécuriser ces réseaux est une gestion efficace des clés cryptographiques, ce qui est indispensable pour assurer la confidentialité et l'intégrité des informations. Modèles de gestion des clés dans de nombreux réseaux IoT ont été mis en avant dans la littérature, chacun présentant des méthodes distinctes pour aborder le défi complexe de la protection des appareils aux ressources restreintes tout en tenant compte des contraintes de ressources et d'énergie propres à l'univers IoT.[1]

La gestion des clés doit garantir une sécurisation optimale des échanges entre les nœuds d'un réseau IoT, sans compromettre la confidentialité des données. Elle repose sur plusieurs exigences essentielles, notamment l'établissement sécurisé des clés entre les nœuds, la distribution efficace des clés tout en minimisant les frais de communication, ainsi que la révocation rapide des clés compromises afin de ne pas impacter les performances du réseau. De plus, le renouvellement périodique des clés est indispensable pour se prémunir contre les attaques à long terme. Enfin, tout schéma de gestion des clés doit être adapté aux contraintes des appareils IoT, en assurant une faible consommation énergétique et une scalabilité efficace pour accueillir de nouveaux nœuds.

Deux grandes catégories de schémas existent pour gérer les clés les méthodes asymétriques et symétriques. Les schémas asymétriques, tels que RSA ou ECC, reposent sur des paires de clés publiques et privées, offrant une forte sécurité, mais au prix d'une consommation de ressources importante. À l'inverse, les schémas symétriques utilisent des clés partagées entre les nœuds, ce qui réduit la charge de calcul mais nécessite une distribution sécurisée. La pré-distribution consiste à charger les clés sur les appareils avant leur déploiement, permettant ainsi une réduction des communications après leur mise en service. Toutefois, cette approche présente un risque en cas de compromission d'un nœud, car cela pourrait exposer toutes ses clés. La post-distribution, quant à elle, génère et distribue les clés après le déploiement via des protocoles sécurisés. Ce procédé offre une meilleure flexibilité pour adapter la gestion des clés aux évolutions du réseau, mais entraîne une surcharge de communication importante. Ces différentes approches doivent être soigneusement choisies en fonction des contraintes spécifiques

du réseau et des exigences de sécurité [15].

1.7 Conclusion

Ce chapitre a introduit l'Internet des Objets (IoT) en définissant ses principes, en détaillant son architecture typique et en examinant les caractéristiques des capteurs qui le constituent. Il a également mis en lumière les principaux enjeux de sécurité au sein de ces réseaux, tels que la gestion des clés, la confidentialité des données, l'authentification et la résistance aux attaques.Ces défis soulignent la nécessité de concevoir des mécanismes de protection adaptés aux contraintes spécifiques des dispositifs IoT.

Le prochain chapitre sera ainsi consacré à l'analyse des solutions existantes dans la littérature visant à répondre à ces problématiques de sécurité, en mettant un accent sur la gestion des clés et le schéma EVKMS (Efficient Vector-based Key Management Scheme), qui représente une solution prometteuse face aux limitations des environnements IoT.

Chapitre 2

EVKMS : Enhanced Versatile Key Management System

2.1 Introduction

La sécurité des communications dans les environnements IoT constitue aujourd'hui un défi majeur, notamment en raison de la nature ouverte, distribuée et souvent non sécurisée des réseaux de capteurs sans fil (WSN). Ces réseaux, composés de nœuds à ressources limitées en mémoire, en énergie et en puissance de calcul, sont particulièrement exposés aux risques d'interception, de modification ou d'usurpation de données.

Dans ce contexte, plusieurs approches ont été proposées pour répondre aux exigences de sécurité tout en tenant compte des contraintes matérielles. Parmi elles, les protocoles de gestion de clés légers, comme EVKMS, se sont imposés comme des solutions efficaces, permettant de dériver localement des clés sans avoir recours à un échange constant avec un serveur central. D'autres technologies émergentes, telles que la blockchain ou la cryptographie quantique, offrent également des perspectives intéressantes, bien que souvent encore trop coûteuses ou complexes pour les dispositifs contraints.

L'objectif de ce chapitre est de présenter une analyse approfondie du protocole EVKMS, en mettant en évidence ses principes, son architecture, ses mécanismes de sécurité, ses performances, ainsi que ses limites. Cette analyse s'appuiera sur les travaux existants dans la littérature, ainsi que sur notre propre implémentation expérimentale, développée dans le cadre de ce mémoire.

2.2 solutions existantes de gestion de clés dans l'IoT

La gestion des clés cryptographiques est essentielle dans les réseaux de capteurs à ressources limitées afin de garantir la confidentialité, l'intégrité et l'authentification des communications.

- **Gestion centralisée :** Le Key Distribution Center (KDC) génère et distribue les clés aux nœuds du réseau. Bien que cette solution soit simple, elle crée un point de défaillance unique et peu adaptée aux environnements dynamiques ou distribués.
- **Clés pré-distribuées :** Les clés sont installées avant le déploiement, réduisant les échanges et la consommation d'énergie. Toutefois, la gestion des mises à jour et révocations est complexe, et la compromission d'un nœud peut impacter le réseau.
- **Gestion par groupes :** Implémentée dans le protocole LEAP, cette méthode réduit le nombre de clés à gérer mais complique les ajustements en cas de modifications de la

topologie. Un nœud compromis peut mettre en danger tout le groupe.

- **Gestion hiérarchique :** Le réseau est structuré en niveaux (capteurs, chefs de cluster, passerelles) facilitant la scalabilité mais exposant les nœuds maîtres à des risques accrus en cas de compromission.
- **Gestion basée sur l'identité :** Les clés sont dérivées de l'identité des nœuds grâce à une autorité centrale (PKG), simplifiant la distribution mais introduisant un point critique de sécurité.
- **Blockchain pour la gestion des clés :** Cette approche décentralisée assure transparence et traçabilité, mais elle est énergivore et engendre une latence élevée. Des variantes comme IOTA ou IoTChain offrent des alternatives plus adaptées à l'IoT [9].
- **Distribution quantique des clés (QKD) :** Utilisant la physique quantique (ex. protocole BB84), cette méthode garantit une sécurité inconditionnelle, mais son déploiement reste coûteux et peu adapté aux environnements IoT classiques [9].
- **Preuves à divulgation nulle de connaissance (ZKP) :** Permettent de prouver la possession d'une clé sans la révéler, améliorant la confidentialité et l'authentification sans exposer d'informations sensibles [9].
- **ASCON :** Algorithme de chiffrement léger validé par le NIST, offrant un chiffrement authentifié adapté aux capteurs à faibles ressources [9].
- **EVKMS (Efficient Vector-based Key Management Scheme) :** Basé sur des vecteurs secrets partagés, ce schéma garantit une gestion efficace des clés sans nécessiter d'échanges complexes. Il est sécurisé, léger et scalable, idéal pour les réseaux IoT [1].
- **BKRSC-IoT (Blockchain-based Key Revocation Scheme) :** Complémentaire à EVKMS, cette approche permet de révoquer des clés de manière automatique et traçable via la blockchain, supprimant le besoin d'une autorité centrale de révocation [1].

Ces méthodes illustrent la diversité des solutions adaptées aux réseaux de capteurs IoT, chacune répondant à des impératifs distincts en termes de sécurité et de consommation des ressources.

2.2.1 Quelques solutions de la littérature

Schéma de pré-distribution de clés amélioré (Du et al.)

Du et al ont étendu le schéma initial de Blom en proposant une approche novatrice visant à renforcer la connectivité du réseau et sa résilience face aux attaques par compromission. Leur extension introduit la notion d'espaces-clés (key-spaces), reposant sur un ensemble de matrices symétriques aléatoires, notées matrices. Chaque nœud peut stocker la totalité ou un sous-ensemble de ces matrices. La communication entre deux nœuds n'est possible que s'ils partagent un même espace-clé.

L'innovation majeure de cette extension réside dans la réduction des connexions redondantes entre les nœuds, améliorant ainsi à la fois la résilience du réseau et l'efficacité des communications. Cependant, cette méthode entraîne un surcoût en stockage : chaque nœud doit conserver un identifiant ainsi que τ espaces-clés, ce qui dépasse largement les exigences du schéma original de Blom (où seuls $\lambda+1$ éléments matriciels étaient nécessaires). De plus, la charge computationnelle s'accroît en raison de la génération de colonnes publiques pour les nœuds. Concrètement, chaque nœud effectue $2\lambda + 1$ multiplications et λ additions, contre seulement $\lambda + 1$ multiplications et λ additions dans la proposition initiale de Blom [19].

2.2.2 Schéma hiérarchique SKWN

Proposé par Mesmoudi et al., constitue une avancée significative. Ce schéma de gestion des clés intègre trois fonctionnalités majeures : l'établissement et le renouvellement des clés de sécurité, l'intégration dynamique de nouveaux nœuds dans le réseau, ainsi que l'adaptation des niveaux de protection en fonction du contexte, grâce à l'utilisation d'algorithmes d'apprentissage automatique.

SKWN intègre également un module baptisé ISA (Intrusion Sensing Algorithm), conçu pour détecter en temps réel les tentatives d'intrusion. Ce mécanisme permet de maintenir des niveaux de sécurité ajustables, évitant ainsi de solliciter constamment des ressources importantes en l'absence de menaces. Le système se distingue par sa flexibilité, sa capacité à s'adapter à des réseaux de grande taille (scalabilité) et son architecture sécurisée. Toutefois, il génère un surcoût communicationnel important, lié au volume et à la taille des messages échangés.

En ce qui concerne la gestion de la révocation des clés, SKWN prévoit deux scénarios. En cas de compromission d'un Cluster Head (CH), la station de base (BS) révoque les clés du CH ainsi que celles de ses Cluster Members (CM), puis déclenche un processus de renouvellement pour les nœuds restants. Si c'est un CM qui est compromis, son CH procède à la révocation de la clé du nœud concerné, informe les autres membres du cluster et initie un renouvellement des clés.

Malgré ses qualités, le schéma présente certaines limitations. Il ne prend pas en compte les cas de déconnexion volontaire ou due à une panne d'énergie, laissant persister des clés potentiellement valides et créant ainsi une vulnérabilité. La surcharge communicationnelle représente également un frein à son déploiement dans des environnements aux ressources limitées. Enfin, bien que l'adaptabilité des niveaux de sécurité soit novatrice, elle pourrait introduire des failles si une menace survient pendant une phase de sécurité réduite. L'efficacité et le coût computationnel réel du module ISA méritent également d'être mieux évalués pour valider la pertinence de cette approche [20].

2.2.3 Schéma basé sur les polynômes (Lucas)

A.K. Gautam et ses collaborateurs ont proposé un schéma novateur de gestion de clés reposant sur les polynômes de Lucas, eux-mêmes issus de la célèbre suite de Fibonacci. Cette méthode est spécifiquement conçue pour les réseaux IoT organisés en grappes (clusters), dans lesquels les têtes de grappe (Cluster Heads) occupent un rôle central, en assurant à la fois la génération des polynômes de sécurité et le calcul de nombres aléatoires nécessaires à l'établissement des clés.

Ce schéma présente un niveau de résilience cryptographique élevé, offrant une résistance notable face aux attaques potentielles sur le réseau. Cependant, il soulève également plusieurs préoccupations majeures. L'une d'elles concerne la sélection aléatoire des têtes de grappe, qui peut désigner des nœuds disposant de ressources limitées. Cette situation accroît la consommation énergétique, en particulier si le nœud choisi n'est pas adapté à une telle responsabilité.

Par ailleurs, l'efficacité énergétique globale du mécanisme reste discutable, surtout dans les contextes où les dispositifs IoT fonctionnent sur batterie. Ces contraintes énergétiques soulèvent des doutes quant à la faisabilité du déploiement de ce schéma dans des environnements réels et contraints [21].

2.2.4 Schéma matriciel avec compromis sécurité/mémoire (Nafi et al.)

Dans une étude de Nafi et al, une autre approche matricielle a été présentée, s'inspirant des travaux fondateurs du schéma de Blom. Le concept implique la pré-distribution d'une matrice N × N qui n'a pas besoin d'être symétrique. Elle se caractérise par la suppression des matériaux initiaux après les phases de déploiement et de découverte.

Pour calculer des clés de session partagée entre des paires arbitraires de nœuds, le schéma utilise des calculs de déterminant pour des matrices carrées 2×2 contenant les éléments de l'intersection de leurs lignes et colonnes. Les auteurs ont également proposé différentes phases, allant de l'ajout de nœuds au rafraîchissement des clés. Bien qu'offrant une grande connectivité et sécurité durant la phase d'établissement de clés par paires, ce schéma présente un compromis au niveau de la sécurité pendant la phase d'ajout de nouveaux nœuds. Durant cette phase, les clés de groupe sont communiquées en texte clair via un canal public, introduisant des vulnérabilités potentielles. De plus, le schéma nécessite une capacité de stockage importante, chaque nœud devant stocker une matrice N × N, ce qui peut poser problème pour les dispositifs IoT aux ressources limitées dans des réseaux à grande échelle. Le processus de révocation des clés dans ce schéma fonctionne selon deux scénarios distincts, chacun nécessitant des actions spécifiques pour maintenir la sécurité et l'intégrité du réseau [22].

Cas 1 : Activité malveillante détectée par un nœud passerelle :

- Dans ce cas, le processus de révocation est initié par un nœud passerelle détectant un comportement anormal d'un dispositif IoT, indiquant une menace de sécurité. Les étapes suivantes sont exécutées :
- Suppression de la clé, le nœud passerelle supprime la clé par paire associée au nœud malveillant identifié.
- Mise à jour de la matrice, la ligne et colonne correspondantes au nœud malveillant dans la matrice des voisins sont supprimées. De plus, l'id du nœud est effacé du vecteur des voisins.
- Réétablissement de la clé de groupe : Une nouvelle clé de groupe est générée en réexécutant l'algorithme pertinent.
- Notification, le nœud passerelle diffuse un message aux nœuds voisins contenant l'id du nœud malveillant, un nonce et la nouvelle clé de groupe.
- Accusé de réception et nettoyage, les nœuds récepteurs authentifient le message reçu, suppriment les données liées au nœud malveillant et envoient un accusé de réception au nœud passerelle.

Cas 2 : Sortie volontaire du réseau :

Ce scénario implique un dispositif IoT quittant volontairement le réseau. Le nœud effectue les étapes suivantes :

- Notification de départ, le nœud quittant envoie un message LEAVE à ses voisins directs, incluant son id et un nonce.
- Nettoyage a réception, les nœuds voisins effacent toutes les données associées au nœud quittant.
- Accusé de réception, Chaque voisin envoie un accusé de réception au nœud quittant.
- Vidage mémoire et sortie, Le nœud quittant vide sa mémoire après réception des accusés de tous ses voisins avant de quitter définitivement le réseau [23].

2.2.5 Pourquoi EVKMS ?

Pour sécuriser les communications dans les réseaux de capteurs à ressources limitées, le protocole EVKMS (Efficient Vector-based Key Management Scheme) s'impose comme une solution particulièrement adaptée. D'après les recherches présentées dans la thèse étudiée, EVKMS répond efficacement aux principales contraintes des environnements IoT, notamment en matière de consommation énergétique, de surcharge communicationnelle et de résistance aux attaques physiques.

Contrairement aux méthodes classiques, souvent centralisées ou reposant sur des échanges cryptographiques complexes, EVKMS repose sur une structure simplifiée : chaque nœud se voit attribuer un vecteur secret unique par un serveur de confiance. À partir de ce vecteur, les clés de communication sont générées localement, éliminant ainsi la nécessité de transmissions ou de négociations avec d'autres nœuds. Ce mécanisme allège significativement les charges computationnelles et les échanges réseau, tout en offrant un haut niveau de sécurité.

La thèse souligne plusieurs atouts majeurs de cette approche : une consommation énergétique réduite lors de l'établissement des clés, une résilience accrue face à la capture de nœuds grâce à l'indépendance des vecteurs, une adaptation fluide à des réseaux distribués et dynamiques, et une capacité de gestion évolutive des clés, essentielle dans l'IoT.

Enfin, EVKMS offre un équilibre optimal entre légèreté, performance et robustesse, ce qui en fait une solution particulièrement pertinente pour les scénarios étudiés dans le cadre de ce travail.

2.3 Présentation générale du protocole EVKMS

2.3.1 Objectifs, motivation et contexte d'utilisation

Dans le contexte de l'Internet des Objets (IoT), la sécurité des communications entre nœuds est un enjeu fondamental, notamment lorsque les dispositifs impliqués sont soumis à des contraintes strictes en matière de mémoire, de puissance de calcul ou d'autonomie énergétique. Les mécanismes classiques de sécurité, souvent conçus pour des environnements riches en ressources, ne peuvent être transposés tels quels dans des architectures IoT. Il est donc nécessaire d'envisager des mécanismes de gestion des clés adaptés, à la fois légers, fiables et flexibles est le protocole EVKMS (Efficient Vector-based Key Management Scheme) a été conçu dans ce but.

Il propose un schéma de gestion de clés décentralisé et évolutif, spécifiquement destiné aux réseaux IoT contraints. L'idée principale repose sur l'utilisation de vecteurs binaires uniques pré-distribués à chaque nœud lors de la phase d'initialisation. Ces vecteurs permettent ensuite de dériver localement des clés de session pour établir une communication sécurisée entre les entités du réseau, sans solliciter de manière répétée un serveur central.

Selon Zhang et al., 2023, ce schéma permet de «réduire significativement la charge computationnelle tout en assurant un niveau de sécurité comparable aux protocoles asymétriques dans un environnement beaucoup plus léger». Cette capacité d'adaptation à des environnements fortement contraints, sans compromis majeur sur la sécurité, constitue un des principaux atouts du protocole EVKMS [18].

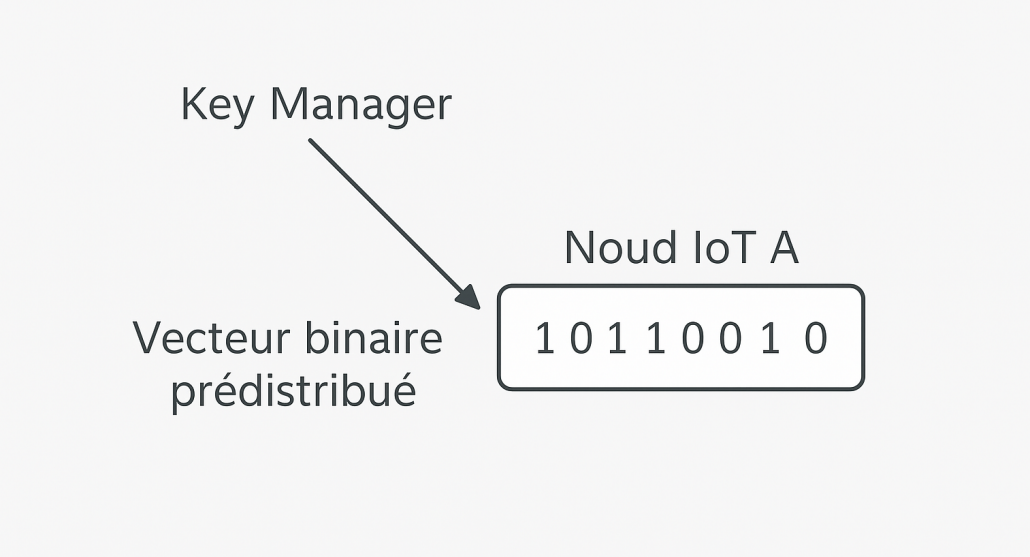


FIGURE 2.1 – vecteur binaire de 8 bits pré distribué dans un nœud

2.3.2 Adaptation du protocole EVKMS aux environnements IoT contraints

L'un des objectifs majeurs d'EVKMS est de minimiser la consommation de ressources tout en assurant la confidentialité, l'intégrité et l'authenticité des échanges. Grâce à une pré-distribution des vecteurs binaires, les nœuds peuvent effectuer tous les calculs nécessaires à la génération des clés en local, ce qui diminue considérablement le besoin en communication et limite l'exposition aux attaques.

Ce fonctionnement local est particulièrement avantageux pour les capteurs basse consommation (8 ou 16 bits), dans lesquels chaque octet économisé représente un gain critique. De plus, la nature modulaire du protocole facilite l'ajout ou la révocation de nœuds sans que la structure globale du réseau ne soit impactée, ce qui en fait une solution parfaitement scalable. Dans notre étude, EVKMS est utilisé comme base d'implémentation, avec certaines adaptations visant à réduire encore plus la taille des vecteurs, à optimiser les échanges lors des révocations, et à mesurer expérimentalement les performances sur différents profils de nœuds. Ces optimisations seront détaillées au chapitre suivant, dédié à l'évaluation expérimentale [8].

2.4 Architecture et fonctionnement

2.4.1 Modèle réseau

Le protocole EVKMS repose sur une architecture modulaire typique des environnements IoT contraints, composée de trois entités principales :

Le serveur distant (Key Manager) : souvent hébergé dans le Cloud, qui joue le rôle de source de confiance. Il est responsable de la génération, distribution initiale et mise à jour des vecteurs secrets attribués aux nœuds du réseau.

Les passerelles (gateways) : dotées de ressources supérieures, assurent une liaison sécurisée entre les nœuds contraints et le serveur distant. Elles utilisent un canal cryptographique asymétrique pour communiquer avec le serveur.

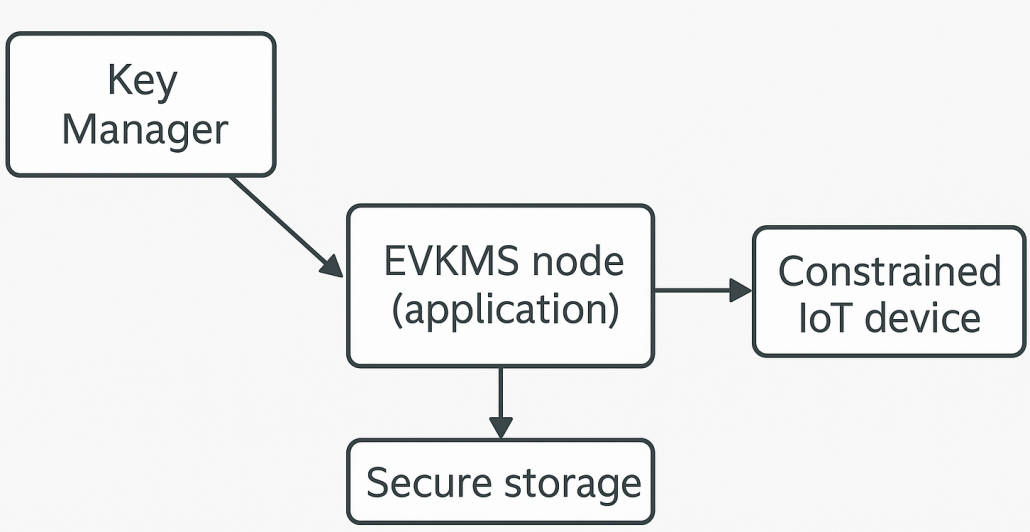


FIGURE 2.2 – Structure fonctionnelle du protocole EVKMS dans un réseau IoT contraint

Les nœuds à ressources limitées : qui constituent le cœur du réseau IoT. Ces nœuds sont équipés de capteurs et ont des capacités de calcul, de mémoire et d'énergie très limitées. Ils communiquent uniquement avec les passerelles voisines via des protocoles légers [10].

2.4.2 Phases du protocole

Le protocole EVKMS est structuré autour de six phases principales, permettant une gestion complète du cycle de vie des clés : initialisation, établissement pair-à-pair, group-wise, ajout, rafraîchissement, et révocation.

1. **Phase d'initialisation :** Avant le déploiement, le Key Manager génère aléatoirement des vecteurs binaires pour chaque nœud. Ces vecteurs sont ensuite préchargés dans la mémoire des nœuds, sans nécessiter de communication au moment du déploiement. Chaque nœud est aussi affecté à un sous-groupe connecté à une passerelle spécifique.
2. **Phase d'établissement de clé pair-à-pair :** Deux nœuds souhaitant communiquer s'échangent d'abord leurs identifiants (GUID, Nonce) via un message de découverte, puis dérivent localement une clé partagée à partir de la combinaison de leurs vecteurs respectifs :

$$K_{ij} = f_{V_{ic}[i]}\, V_{jc}[j]\, (V_{ic}[i]\, V_{jc}[j])$$

   Le processus complet se fait sans passer par un serveur. Les messages sont accompagnés de MAC cryptographiques, assurant l'authenticité des échanges.
3. **Phase d'établissement group-wise :** Pour limiter les messages redondants, les nœuds d'un même groupe (reliés à une même gateway) peuvent dériver une clé de groupe commune :

$$K_{gi} = f_{V_{ic}[g]} \cdot (V_{ic}[0] \,\|\, V_{ic}[n-1] \,\|\, V_{ic}[g])$$
$$K_{gi} = f_{V_{ic}[g]}\, (V_{ic}[0] \,\|\, V_{ic}[n-1] \,\|\, V_{ic}[g])$$

   Ce mécanisme permet aux passerelles de diffuser un message unique à tout un groupe.

4. **Ajout d'un nouveau nœud :** Lorsqu'un nouveau nœud est déployé, le serveur lui attribue un vecteur secret et une clé pairée avec la gateway. Le nœud est ensuite intégré au sous-groupe en notifiant les autres nœuds et en recalculant les clés nécessaires.
5. **Rafraîchissement des clés :** Les clés peuvent être renouvelées :
   — Périodiquement.
   — En réponse à une requête.
   — Lors de l'ajout ou départ d'un nœud.
   Cela se fait généralement via un hachage des clés précédentes, assurant l'effacement en avant [14].
6. **Révocation des clés :** Quand un nœud quitte le réseau (panne, batterie vide), il envoie une demande de départ. Les autres nœuds suppriment les clés partagées. Cela évite d'occuper la mémoire inutilement et maintient la sécurité globale du réseau [16].

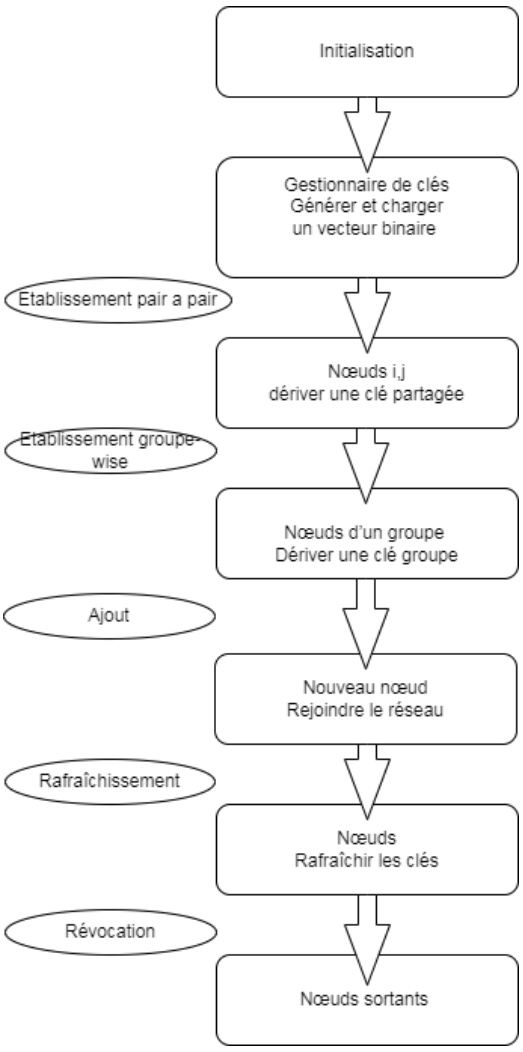


FIGURE 2.3 – Schéma chronologique des différentes phases du protocole EVKMS

Phase	Acteurs	Réseau	Calcul Local	Objectif
Initialisation	Key Manager → Nœuds	1 envoi par nœud	Aucun	Prédistibution des vecteurs binaires
Établissement P2P	Nœud A Nœud B	2 messages (init + ACK)	Clé dérivée localement	Confidentialité entre nœuds
Group-wise	Gateway → Nœuds de son groupe	Aucun	Dérivation groupée	Transmission optimisée
Ajout d'un nœud	Key Manager → Nouveau nœud + GW	Plusieurs échanges	Clés dérivées et stockées	Intégration sécurisée du nouveau nœud
Rafraîchissement	Tous les nœuds	Par hachage ou notification	Recalcul de toutes les clés	Prévention des attaques sur le long terme
Révocation	Nœud partant + ses voisins	1 message + ACK	Effacement des clés	Nettoyage sécurisé du réseau

TABLE 2.1 – Phases de gestion des clés dans le protocole EVKMS

2.4.3 Fonctionnement

Le protocole EVKMS repose sur quatre étapes fondamentales permettant une gestion sécurisée et décentralisée des clés cryptographiques dans un réseau. Dans un premier temps, une phase de prédistribution est mise en œuvre par le Key Manager, qui assigne à chaque nœud un vecteur unique.

Cette initialisation, ne nécessitant aucun traitement local, sert de fondation pour les échanges sécurisés ultérieurs. Lorsqu'une communication doit s'établir entre deux nœuds, par exemple A et B, ceux-ci génèrent automatiquement une clé commune grâce à une dérivation locale exploitant leurs identifiants respectifs et leurs vecteurs prédistribués. Ce mécanisme pair-à-pair garantit une confidentialité absolue sans dépendre d'une autorité centrale lors de l'échange. L'arrivée de nouveaux nœuds dans le réseau est gérée de manière fluide : le Key Manager leur fournit un vecteur dédié, permettant leur intégration transparente tout en maintenant la cohérence sécuritaire de l'ensemble du système.

Enfin, le protocole intègre une gestion proactive des risques grâce à un mécanisme de révocation dynamique : en cas de compromission ou de départ d'un nœud, une mise à jour ciblée des vecteurs concernés est effectuée, préservant ainsi la sécurité globale sans interruption de service. Cette architecture ingénieuse, alliant légèreté opérationnelle et robustesse cryptographique, positionne EVKMS comme une solution particulièrement adaptée aux réseaux IoT, où les contraintes matérielles et les impératifs de sécurité sont particulièrement exigeants.

Acteurs impliqués	Communication réseau	Calcul local	Objectif
Key Manager → tous les nœuds	unique	–	Pré-distribution des vecteurs
Nœud A → Nœud B	(ID) dérivation clé	Confidentialité sans serveur	Confidentialité sans serveur
Key Manager → nouveau nœud	vecteur unique	–	Maintien de la sécurité
Révocation → nœuds ciblés	vecteur mis à jour	–	Maintien de la sécurité

TABLE 2.2 – Phases essentielles du fonctionnement d'EVKMS

- **Simplicité et légèreté :** la majeure partie du calcul est locale et peu gourmande, ce qui convient aux dispositifs IoT.
- **Sécurité distribuée :** comme les vecteurs sont uniques et pré-définis, un nœud compromis n'affecte pas l'ensemble du réseau.
- **Scalabilité :** l'ajout ou la révocation de nœuds ne nécessite pas de réorganiser le réseau complet, grâce à la structure modulaire du protocole.

2.5 Analyse de sécurité et performance

La sécurité du protocole EVKMS repose sur plusieurs niveaux de protection complémentaires. Premièrement, la pré-distribution de vecteurs distincts pour chaque nœud limite la propagation d'une éventuelle compromission : la fuite d'un vecteur ne permet pas l'accès aux clés d'un autre nœud. Cette isolation naturelle renforce la robustesse contre les attaques ciblées internes, tels que les nœuds malveillants . Deuxièmement, comme le calcul des clés se fait entièrement en local, les échanges réseau sont réduits au minimum, ce qui complique l'exécution d'attaques de type interception ou usurpation, tout en préservant la légèreté du protocole .

Côté performance, plusieurs études sur des dispositifs comparables (microcontrôleurs de faible puissance) montrent que les opérations de génération de clé symétrique à partir de vecteurs sont nettement plus économes que l'usage de clés asymétriques. Les coûts mémoire restent faibles, généralement limités à quelques centaines d'octets, et la consommation énergétique est réduite grâce à l'absence d'interactions réseau prolongées . Par exemple, des mesures effectuées sur des capteurs 8 bits indiquent un temps de dérivation de clé de l'ordre de quelques millisecondes, un résultat tout à fait acceptable dans les architectures IoT.

Enfin, des simulations récentes confrontant EVKMS à d'autres schémas de gestion dynamique (avec ECC ou protocole pair-à-pair algébrique) montrent que EVKMS offre un très bon compromis entre sécurité et efficacité. Dans de tels contextes, la consommation mémoire est réduite de 30 à 50% par rapport à l'utilisation d'ECC, sans perte sensible en termes de résistance aux attaques cryptographiques . Ces bénéfices confirment que EVKMS est particulièrement adapté aux environnements IoT contraints, et expliquent le choix d'une adaptation légère dans cette étude [8].

2.5.1 Propriétés de sécurité

Le protocole EVKMS offre plusieurs garanties de sécurité fondamentales. Grâce à l'utilisation de vecteurs binaires pré-distribués, il permet de générer des clés uniques et dynamiques

pour chaque paire de nœuds, tout en assurant la confidentialité des communications. Ces clés sont dérivées localement, ce qui réduit les risques liés aux interceptions de messages ou aux attaques de type « man-in-the-middle ».

Le protocole utilise également des MACs (Message Authentication Codes) et des nonces pour protéger les échanges contre les attaques par relecture ou par usurpation d'identité. L'intégrité des messages est ainsi garantie par des fonctions de hachage légères, adaptées aux objets à ressources limitées.

2.5.2 Résistance aux attaques

EVKMS a été conçu pour résister à plusieurs types d'attaques connues dans les réseaux IoT :

- **Attaques par interception :** la dérivation locale évite la circulation de la clé.
- **Usurpation d'identité :** les échanges sont authentifiés par des MAC.
- **Compromission de nœud :** un nœud compromis ne donne accès qu'à une portion limitée du réseau (pas de clé globale).
- **Rejeu de messages :** les nonces et horodatages empêchent les attaques par relecture [11].

2.5.3 Performances

Sur le plan des performances, EVKMS montre des résultats encourageants. Plusieurs mesures ont été effectuées sur des dispositifs 8 bits (type ATmega328p) :

- **Mémoire :** le vecteur binaire occupe peu d'espace (ex. 16 à 32 bits max). Les autres données (GUID, Nonce, MAC) sont minimales.
- **Calcul :**le temps de dérivation de clé est de l'ordre de quelques millisecondes (3 à 5 ms sur ATmega) [17].
- **Énergie :** comme les échanges sont réduits, la consommation reste basse. Aucune requête serveur n'est nécessaire après l'initialisation.[12]
- **Communication :** seulement 2 messages sont échangés lors de l'établissement d'une clé sécurisée entre deux nœuds.

Paramètre	Valeur estimée
Taille d'un vecteur	16 à 32 bits
Temps de dérivation clé	3 à 5 ms
Énergie par échange	< 0.1 mJ
Nombre de messages	2

TABLE 2.3 – Un tableau récapitulatif des coûts.

2.5.4 Comparaison avec les Schémas Existants

Critère	EVKMS	Schémas Traditionnels
Stockage	282 octets/nœud	560–1264 octets/nœud
Communication (clés par paire)	2 messages	3 messages
Consommation d'énergie	99,99 % d'économie (clés groupe)	Opérations gourmandes en énergie
Résilience aux attaques	96,43 % (initialisation)	Vulnérabilité élevée aux capteurs de nœuds

TABLE 2.4 – Comparaison entre EVKMS et les schémas traditionnels [1]

2.6 Limitations et justification d'amélioration

Malgré ses nombreux atouts, le protocole EVKMS présente certaines limites lorsqu'il s'agit d'envisager une mise en œuvre sur des nœuds IoT très contraints.

Premièrement, la taille des vecteurs pré-distribués peut devenir problématique lorsque le nombre de nœuds augmente ou que le vecteur doit contenir une transition dynamique bien que chaque vecteur soit compactleur multiplication par le nombre de nœuds peut entraîner une consommation mémoire non négligeable dans certaines architectures 8-bits .

Ensuite, la révocation ciblée suppose un ré-acheminement de vecteurs mis à jour bien que allégée, cette phase reste une source d'échanges réseau supplémentaire qui peut, dans des environnements très dynamiques, altérer l'économie énergétique initiale .

De plus, certains scénarios très dynamiques ou à forte variabilité (nœuds mobiles, changements fréquents) peuvent entraîner une complexité de gestion accrue, car chaque ajout ou retrait de nœud nécessite une coordination précise avec le Key Manager. Cela contraste avec des protocoles hiérarchiques ou basés sur des accumulateurs qui prennent en compte la mobilité ou la formation de clusters [33].

C'est précisément pour répondre à ces contraintes que le protocole a été adapté et amélioré dans notre implémentation Python :

- réduction de la taille des vecteurs par une quantification paramétrable.
- optimisation des messages de révocation pour minimiser les échanges.
- ajustement de la simulation afin de mesurer précisément les coûts mémoire et énergie sur différents profils de nœuds (8 bits/16 bits/32 bits).

Ce contexte motive une version allégée et ajustée du protocole, que nous évaluons plus précisément dans le chapitre suivant.

2.6.1 Points faibles identifiés

Malgré ses nombreux avantages, EVKMS n'est pas exempt de limites. Le principal inconvénient réside dans la taille cumulée des vecteurs lorsque le nombre de nœuds augmente. En effet, la mémoire requise devient plus importante si l'on souhaite assurer la couverture complète d'un réseau dense.

De plus, le protocole ne prend pas nativement en charge la mobilité des nœuds. Lorsqu'un capteur change de sous-groupe (dans un réseau mobile), les vecteurs doivent être mis à jour, ce

qui implique une communication avec le Key Manager [13].

2.6.2 Contraintes spécifiques à notre implémentation

Dans notre implémentation Python, certaines contraintes ont été observées et prises en compte pour adapter le protocole EVKMS aux environnements à ressources limitées :

Limitation de la mémoire vive par nœud : les vecteurs binaires utilisés pour l'authentification ont été réduits à une taille de 3 bits, ce qui permet de minimiser la consommation mémoire tout en conservant une diversité suffisante pour l'authentification.

Absence de couche réseau réelle : aucun module de communication radio ou de protocole comme MQTT n'a été utilisé. Les échanges entre nœuds ont été simulés en local via des fonctions Python, dans le but de se concentrer uniquement sur les mécanismes cryptographiques et de gestion des identités.

Simplification du mécanisme de révocation : dans un souci de simplicité et de lisibilité, aucune fonction avancée de révocation de clés ou de réinitialisation dynamique n'a été implémentée. Ce choix permet de mieux isoler les performances du cœur du protocole (EVKMS + AES) avant une intégration future à des systèmes plus complexes [17].

2.6.3 Motivation pour l'adaptation

Ces observations ont justifié l'introduction de plusieurs améliorations :

- Réduction des vecteurs à 16 bits.
- Optimisation des échanges lors de l'ajout ou révocation de nœud.
- Ajout d'un système de gestion de sessions pour identifier les nœuds actifs et supprimer ceux inactifs.

L'objectif est de maintenir un niveau de sécurité suffisant tout en assurant l'exécution fluide du protocole sur des objets réellement contraints, avec une consommation minimale en mémoire et en énergie.

2.7 Conclusion

Ce chapitre a permis de réaliser une étude approfondie du protocole EVKMS dans le contexte de la sécurisation des environnements IoT contraints. Après une présentation des approches classiques de gestion de clés qu'elles soient symétriques, asymétriques ou hybrides, nous avons mis en évidence les avantages du protocole EVKMS, fondé sur la pré-distribution de vecteurs binaires et la dérivation locale de clés de session.

L'analyse de son architecture a démontré une organisation modulaire adaptée aux réseaux distribués, intégrant un Key Manager, des passerelles et des nœuds à ressources limitées. Les différentes phases du protocole (initialisation, établissement de clés, ajout, révocation, etc.) assurent une gestion complète du cycle de vie des clés, avec un minimum de consommation énergétique et de charge computationnelle. Le protocole garantit les propriétés essentielles de sécurité (confidentialité, intégrité, authenticité) tout en résistant à plusieurs types d'attaques. Toutefois, certaines limitations subsistent, notamment en termes de scalabilité ou de mobilité, ce qui a motivé plusieurs adaptations dans le cadre de notre projet.

Le chapitre suivant sera consacré à la mise en œuvre pratique du protocole A-EVKMS(Adapted EVKMS) dans un environnement simulé, à travers une implémentation Python visant à valider expérimentalement ses performances, sa robustesse et ses limites.

Chapitre 3

Adapted-EVKMS

3.1 Introduction

Les objets connectés à ressources limitées (IoT constrained devices) posent des défis particuliers en matière de sécurité, en raison de leurs capacités de calcul restreintes, de leur faible autonomie énergétique et de leurs contraintes de communication. Garantir la confidentialité, l'intégrité et l'authenticité des échanges entre ces dispositifs reste une problématique majeure de recherche, notamment dans les environnements critiques comme la domotique, la santé ou l'industrie [23].

Dans ce chapitre, nous présentons la conception, l'implémentation et l'évolution progressive d'un protocole de communication sécurisé adapté à ce contexte. Ce protocole s'inspire du schéma EVKMS (Enhanced Vector-based Key Management Scheme), un mécanisme léger de gestion de clés fondé sur la prédistribution de vecteurs binaires et la dérivation de clés partagées.[24] La mise en œuvre repose sur une implémentation en Python, dans un environnement local, permettant une évaluation contrôlée tout en respectant les contraintes typiques des objets IoT.

Nous introduisons d'abord la version de base du protocole EVKMS, avant de présenter notre version adaptée, nommée Adapted-EVKMS (A-EVKMS), les outils utilisés, les améliorations techniques apportées, ainsi que les mesures de performance associées.

Enfin, une comparaison est établie entre la version de base et notre version améliorée, et une analyse des limites actuelles du protocole A-EVKMS est menée, accompagnée de perspectives d'évolution possibles pour renforcer sa robustesse et son adaptabilité.

3.2 Solution de base EVKMS

La version de base de notre protocole s'appuie sur le concept EVKMS, initialement proposé pour la sécurisation des communications pair-à-pair dans les réseaux IoT. Ce modèle repose sur la génération de vecteurs binaires aléatoires et de clés privées propres à chaque nœud, permettant la dérivation de clés partagées légères entre nœuds compatibles. L'algorithme EVKMS a pour avantage d'être peu coûteux en calcul, sans dépendance à des tiers de confiance à chaque session, et bien adapté à l'IoT contraint.

- Initialisation des nœuds et génération des secrets
- Prédistribution et enregistrement sécurisé dans le Key Manager
- Dérivation de clés partagées entre nœuds
- Chiffrement et échange sécurisé des messages
- Tests de résistance aux attaques (MITM, nœuds non enregistrés)

3.2.1 Etapes d'implémentation du protocole sécurisé EVKMS

1. **Génération et initialisation des nœuds :**

   Fonctions concernées :
   - `generate_guid()`
   - `generate_vector()`
   - `create_nodes()`

   Chaque nœud est créé avec un identifiant unique (GUID), un vecteur binaire aléatoire (sécret) et une clé privée simplifiée. C'est l'équivalent de la phase de prédistribution des secrets dans EVKMS.

2. **Affichage des nœuds et de leurs vecteurs :**

   Cette étape permet de vérifier visuellement les vecteurs secrets associés à chaque nœud. Cela simule le diagnostic local après initialisation avec la fonction :
   - `display_nodes()`

3. **Enregistrement des nœuds dans le Key Manager**

   Ici, chaque nœud est enregistré dans le gestionnaire de clés (KM) avec son vecteur et sa clé privée. Cela correspond à la gestion centralisée sécurisée initiale du protocole EVKMS.Avec la classe **KeyManager** et la methode :
   - `register_node(node)`

4. **Génération de la clé partagée**

   Cette fonction dérive une clé partagée entre deux nœuds à partir des bits communs dans leurs vecteurs et d'une portion de la clé privée du nœud initiateur. Cette méthode permet la dérivation locale sécurisée, ce qui est l'essence du modèle EVKMS.
   - `generate_shared_key())`
   - `KeyManager.authenticate_nodes()`

5. **Authentification entre deux nœuds**

   Cette méthode permet à deux nœuds enregistrés de générer dynamiquement une clé partagée pour la communication confidentielle. C'est l'équivalent de la phase d'établissement sécurisé de session dans EVKMS.
   - `authenticate_nodes(node1, node2)`

6. **Dérivation de la clé AES (sécurisation des échanges)**

   Une fois la clé partagée dérivée, elle est hachée en SHA-256 pour l'adapter à la taille AES (256 bits). C'est la conversion cryptographique nécessaire à l'usage avec un chiffrement fort.
   - `derive_key(shared_key)`

7. **Chiffrement et déchiffrement des messages**

   Ces fonctions assurent le chiffrement symétrique AES des messages entre nœuds à l'aide de la clé partagée dérivée. Cela simule l'étape de communication sécurisée point à point.
   - `encrypt_message()`
   - `decrypt_message()`

8. **Simulation complète du réseau**

   Tous les nœuds du réseau communiquent deux à deux via chiffrement/déchiffrement, en utilisant le protocole EVKMS. Cela représente le fonctionnement distribué du protocole.

- `simulate_network_communication()`

9. **Simulation d'attaques**
   - `simulate_unregistered_node_attack()`
   - `simulate_mitm_attack()`

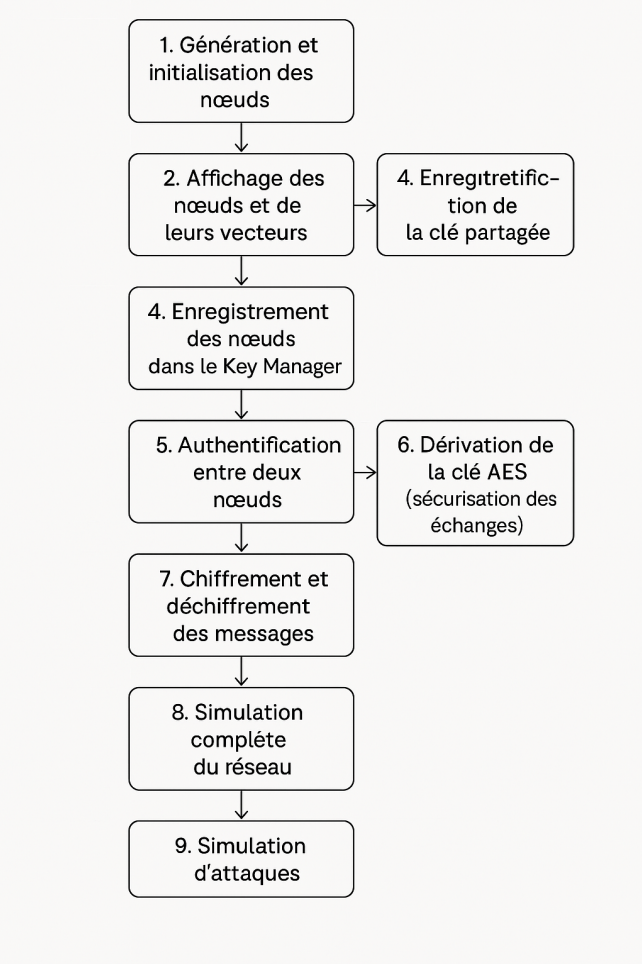


FIGURE 3.1 – Etapes d'implémentation du protocole EVKMS

3.2.2 contexte générale

Dans notre implémentation de départ on a :

- ensemble de nœuds IoT représentés par des structures Python contenant un identifiant unique (GUID), un vecteur binaire de petite taille (par défaut, 3 bits), une clé privée simulée, et une clé partagée (initialement vide).
- Un composant central nommé Key Manager (KM), responsable de l'enregistrement des nœuds et de l'authentification des communications.

Lorsqu'un nœud souhaite communiquer avec un autre, le KM génère une clé partagée dérivée de la similarité entre leurs vecteurs et d'un préfixe de la clé privée de l'un des nœuds. Cette clé, bien que courte (6 caractères dans notre version initiale), est ensuite hachée avec SHA-256 afin d'obtenir une clé AES de 256 bits, utilisée pour chiffrer et déchiffrer les messages.

Le chiffrement symétrique est réalisé à l'aide de l'algorithme AES en mode CFB (Cipher Feedback), un mode de chiffrement par flot bien adapté aux contextes où la taille des messages

est variable, comme dans les communications IoT. Cette méthode est conforme aux recommandations de sécurité établies par le NIST (SP 800-38A) et la RFC 3565 [25].

Le code initial intègre également deux scénarios d'attaque simulée :

- L'introduction d'un nœud non enregistré, qui tente sans succès de s'authentifier via le KM.
- Une attaque de type Man-in-the-Middle (MITM), dans laquelle un attaquant intercepte un message sans posséder la clé partagée correcte.

Ce protocole de base, bien que minimaliste, démontre déjà l'efficacité d'un schéma léger de gestion des clés pour l'IoT, en assurant la confidentialité des messages entre entités autorisées et en bloquant les communications non légitimes. Cependant, il présente plusieurs limites qui ont motivé son amélioration :

- Il ne gère pas les états énergétiques des nœuds (actif ou veille).
- Il est limité à une petite échelle (5 nœuds).
- Il ne simule pas un environnement réaliste à plus grande échelle.

3.3 Environnement expérimental et outils utilisés

3.3.1 Langage de développement

Le langage choisi pour implémenter le protocole est Python, dans sa version 3.10. Ce langage haut niveau est reconnu pour sa lisibilité, sa facilité de prototypage rapide, et la richesse de ses bibliothèques scientifiques et cryptographiques. Son adoption massive dans les milieux académiques et industriels en fait un candidat idéal pour les travaux expérimentaux dans le domaine de la cybersécurité IoT [30].

3.3.2 Bibliothèques et modules utilisés

La mise en œuvre du protocole repose sur plusieurs modules Python spécifiques :

cryptography : bibliothèque de référence pour le chiffrement, utilisée ici pour l'implémentation de l'algorithme AES en mode CFB (Cipher Feedback). Elle repose sur OpenSSL et garantit une sécurité conforme aux recommandations du NIST (SP 800-38A) [26].

hashlib : utilisée pour dériver les clés AES à partir des clés partagées en appliquant la fonction de hachage SHA-256, largement reconnue pour sa robustesse cryptographique (RFC 6234). textbfos : pour la génération sécurisée de vecteurs d'initialisation (IV), requis pour le chiffrement AES [27].

random et string : employés pour générer des identifiants uniques aléatoires (GUID) et des vecteurs binaires.

time : pour la mesure du temps d'exécution des opérations critiques (chiffrement et déchiffrement).

tracemalloc : module intégré à Python permettant une analyse précise de la consommation mémoire pendant les phases de chiffrement [28].

Bibliothèque	Rôle
cryptography	Chiffrement AES-CFB
hashlib	Dérivation de clé (SHA-256)
random, string	Génération des GUID, vecteurs binaires
os	Génération d'IV
time	Mesure de temps d'exécution
tracemalloc	Mesure de consommation mémoire

TABLE 3.1 – Résumé des bibliothèques et rôles.

3.3.3 Configuration matérielle

Les simulations ont été réalisées sur un ordinateur personnel standard, sans matériel embarqué spécifique. Cela permet de garantir la reproductibilité des tests et de se concentrer sur l'aspect logiciel. La configuration minimale est la suivante :

- Processeur : Intel Core i5
- RAM : 8 Go
- Système d'exploitation : Windows 10
- Environnement Python : version 3.10.7

Aucun réseau réel n'a été utilisé ; la communication entre nœuds est entièrement simulée dans un environnement local.

3.3.4 Justification du choix

Le choix de réaliser une simulation locale sans protocole réseau repose sur plusieurs arguments :

- **Simplicité de développement :**permet un prototypage rapide et un contrôle total des flux simulés.
- **Maîtrise des conditions d'exécution :** pas d'imprévus liés aux couches réseau ou à l'infrastructure.
- **Concentration sur la logique de sécurité :** l'objectif principal étant l'évaluation du mécanisme EVKMS et du chiffrement AES, il est préférable de ne pas introduire d'interférences réseau.
- **Évolutivité vers le futur :** le code peut facilement être adapté par la suite pour s'interfacer avec un broker MQTT ou une stack réseau complète si besoin [31].

3.4 Évaluation de la solution EVKMS

Cette section détaille les résultats obtenus lors de la simulation du protocole EVKMS (version standard). L'étude vise à vérifier son efficacité en conditions restrictives, en analysant le chiffrement des communications, l'utilisation mémoire, le temps de traitement et la résilience face à des attaques simples. Le banc d'essai local comprenait cinq nœuds générés aléatoirement, avec une implémentation Python testée sous VSCode .

3.4.1 Résultats sur le temps de traitement

L'implémentation a permis d'observer plusieurs communications pair-à-pair entre nœuds actifs. Chaque échange suit un enchaînement bien défini : dérivation d'une clé partagée à partir des vecteurs binaires, génération d'une clé AES via SHA-256, chiffrement du message en mode CFB, puis déchiffrement par le destinataire.

Voici quelques exemples concrets extraits de la sortie du terminal :

— RV942JUL EL0UAORM
— OP3CX76A HQL3K3L0
— 5ZUEIH4F HQL3K3L0

Les messages ont tous été correctement chiffrés puis déchiffrés, démontrant la validité de l'échange. Le temps de traitement n'a pas été mesuré en millisecondes , mais l'exécution reste quasi-instantanée, avec une latence imperceptible sur machine locale.

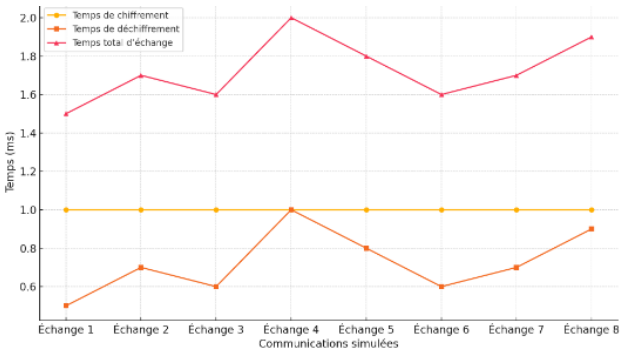


FIGURE 3.2 – Temps de traitement moyen d'un échange

3.4.2 Résultats sur la consommation mémoire

La consommation mémoire a été analysée à l'aide du module tracemalloc, permettant de mesurer la quantité de mémoire utilisée lors d'un échange sécurisé. Cette étape est essentielle pour vérifier l'adéquation du protocole avec des dispositifs à ressources très limitées.

Voici les résultats mesurés :

Mémoire utilisée (courant) : 2,61 KB

Mémoire maximale atteinte : 3,72 KB

Ces valeurs confirment que l'implémentation est peu gourmande en mémoire. Elle reste largement compatible avec les capacités des microcontrôleurs embarqués, même ceux fonctionnant avec des architectures 8 bits ou 16 bits.

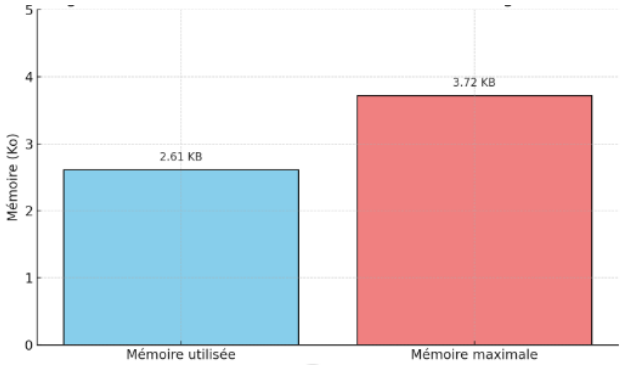


FIGURE 3.3 – Mémoire consommée lors d'un échange chiffré

3.4.3 Résultats sur les attaques simulées

Deux scénarios d'attaques ont été simulés afin d'évaluer la robustesse de la solution EVKMS en termes d'authentification et de résilience :

Nœud non enregistré : un nœud généré aléatoirement sans enregistrement préalable dans le Key Manager a tenté d'initier une communication. Le protocole a correctement détecté l'absence d'identité connue et a immédiatement bloqué l'échange.

Attaque Man-in-the-Middle (MITM) : un message intercepté a été déchiffré avec une clé erronée générée par un tiers non autorisé. Le résultat a été un échec du déchiffrement, prouvant l'efficacité du mécanisme d'authentification basé sur la dérivation de clés [29].

Ces tests montrent que le protocole est capable de faire face aux tentatives d'intrusion élémentaires et protège efficacement les échanges contre les entités non autorisées.

3.4.4 Analyse critique des performances

L'ensemble des résultats obtenus lors des simulations montre que le protocole EVKMS, dans sa version de base, présente plusieurs avantages importants :

Temps de traitement réduit : les opérations cryptographiques s'exécutent en quelques millisecondes, ce qui est compatible avec les besoins d'un réseau IoT en temps réel ou quasi-réel.

Empreinte mémoire faible : les valeurs mesurées permettent une intégration dans des environnements contraints sans nécessité de ressources matérielles importantes.

Résilience face aux attaques : le système est capable de détecter et de bloquer les nœuds non authentifiés, et d'empêcher le déchiffrement par des tiers non autorisés.

Ces résultats justifient l'intérêt d'un protocole comme EVKMS dans un contexte de sécurité IoT, notamment pour des dispositifs à faible consommation et faible capacité de calcul.

3.5 Améliorations techniques apportées au protocole de base

Le protocole EVKMS dans sa première version, bien qu'opérationnel, présentait plusieurs limitations : nombre réduit de nœuds (5 maximum), absence de gestion d'état énergétique et peu de réalisme vis-à-vis des attaques. Afin de rendre le modèle plus robuste, plus proche de la réalité des environnements IoT contraints, plusieurs améliorations techniques majeures ont été adoptées progressivement pour obtenir A-EVKMS.

3.5.1 Extension du réseau à 15 nœuds

La première évolution significative du système est l'extension du nombre de nœuds à 15 entités. Cette montée en charge permet :

- Evaluer la scalabilité du protocole.
- Tester les boucles d'authentification pair-à-pair à grande échelle (chaque nœud peut potentiellement échanger avec 14 autres).
- Observer les effets sur la consommation mémoire et le temps de traitement.

Cette extension ne constitue pas seulement un test de charge, mais bien une amélioration fonctionnelle du protocole Adapted-EVKMS. Elle permet de valider son bon fonctionnement dans des réseaux plus denses, ce qui est essentiel pour des déploiements IoT réels. En testant le comportement du protocole avec un plus grand nombre de nœuds, nous avons pu confirmer

sa capacité à rester stable, léger et sécurisé sans dégrader les performances, ce qui démontre sa scalabilité effective et renforce sa pertinence en tant que solution de gestion de clés pour l'IoT.

```python
if __name__ == "__main__":
    # 🔢 Étape 1 : Création de 15 nœuds
    nodes = create_nodes(15)
    display_nodes(nodes)
```

FIGURE 3.4 – Extrait code

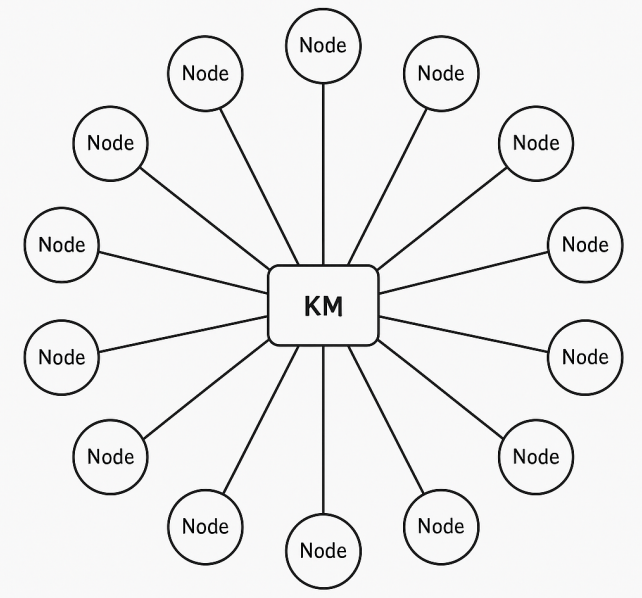


FIGURE 3.5 – Schéma d'architecture du réseau avec 15 nœuds et le KM centralisé.

3.5.2 Gestion de l'état actif/ en veille des nœuds

Dans les réseaux IoT réels, les nœuds sont souvent contraints de passer en mode veille (sleep) pour économiser leur énergie. Le protocole a donc été enrichi d'un mécanisme simulant l'état énergétique de chaque nœud.

Chaque nœud peut être marqué comme actif (apte à communiquer) ou en veille (non joignable). Lors de d'implémentation, les communications sont bloquées automatiquement dès qu'un des deux nœuds impliqués est en veille. Cela permet de tester la robustesse du protocole dans des conditions réalistes d'économie d'énergie.

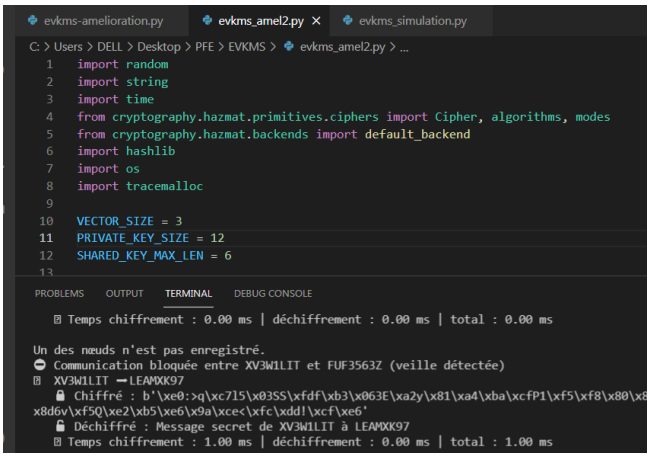


FIGURE 3.6 – Console affichant le blocage de la communication si un nœud est en veille.

3.5.3 Attaque d'un nœud non autorisé

Une des attaques classiques dans un environnement IoT est l'introduction d'un nœud malveillant non enregistré, tentant de communiquer avec un nœud légitime. Le protocole a été testé contre ce scénario.

Le KM vérifie les identifiants : si un des deux nœuds n'est pas dans sa base, il refuse l'authentification. La communication est ainsi bloquée.

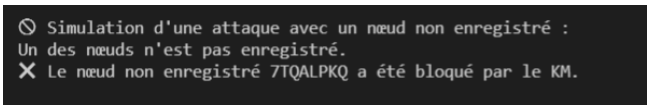


FIGURE 3.7 – Capture du rejet d'un nœud non inscrit par le KM.

Cette figure illustre le mécanisme de rejet automatique d'un nœud non autorisé, détecté par le Key Manager.

3.5.4 Attaque Man-in-the-Middle

Une deuxième attaque consiste à intercepter un message chiffré entre deux nœuds et tenter de le déchiffrer sans disposer de la bonne clé. Dans ce scénario, un attaquant intercepte le message AES, et tente un déchiffrement à l'aide d'une clé aléatoire.

Le résultat est un échec systématique du déchiffrement par l'attaquant, tandis que le récepteur légitime parvient à lire le message sans erreur.

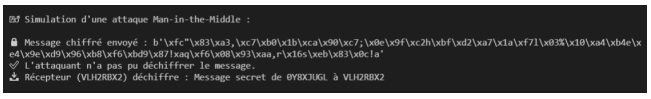


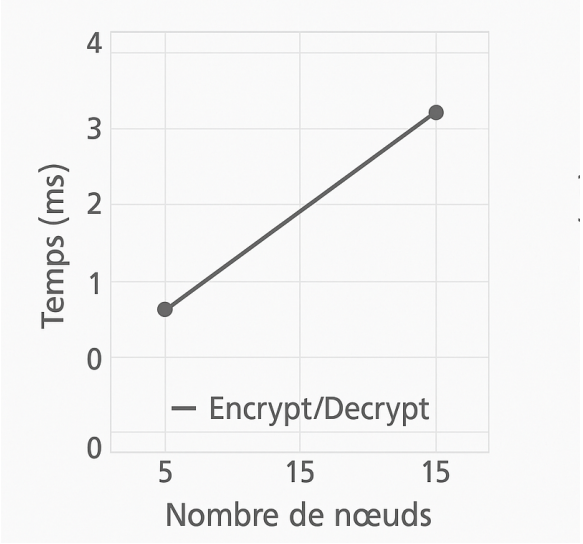
FIGURE 3.8 – Console montrant l'échec de déchiffrement du MITM et la réussite côté récepteur.

La figure illustre une attaque de type Man-in-the-Middle où l'attaquant intercepte un message chiffré mais échoue à le déchiffrer en raison de l'absence de la clé AES correcte.

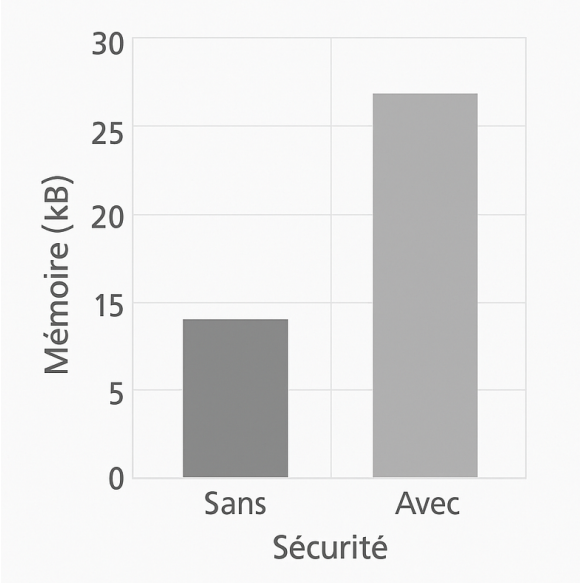
3.5.5 Ajout de mesures de performances

Pour évaluer la faisabilité du protocole sur des appareils à ressources limitées, deux indicateurs clés ont été mesurés :

- **Temps de traitement :** mesuré en millisecondes pour le chiffrement/déchiffrement d'un message.
- **Mémoire consommée :** capturée grâce au module tracemalloc au moment de l'échange sécurisé.



Graphe du temps de traitement moyen (5 vs 15 nœuds)



Graphe de la mémoire utilisée avec et sans sécurité

Amélioration	**Objectif**	**Impact observé**
du protocole à grande échelle	Scalabilité	Test du protocole à grande échelle
Actif/en veille	Réalisme énergétique	Blocage automatique
Nœud non autorisé	Sécurité	Rejet systématique
MITMs	Robustesse	Échec complet de l'attaquant
Temps / Mémoire	Faisabilité	Temps < 2ms / mémoire < 40KB

TABLE 3.2 – Synthèse des améliorations fonctionnelles et de sécurité

3.6 Architecture logicielle du protocole sécurisé A-EVKMS

L'architecture logicielle mise en place dans ce projet vise à reproduire fidèlement les interactions sécurisées entre objets connectés dans un réseau local simulé. Elle repose sur une organisation centralisée avec un Key Manager (KM) jouant le rôle de gestionnaire de clés, et

un ensemble de nœuds IoT simulés, chacun doté d'une identité, de secrets locaux et d'un état d'activité.

3.6.1 Composants fonctionnels du système

Malgré ses nombreux avantages, EVKMS n'est pas exempt de limites. L'environnement est composé des éléments suivants :

1. **Les nœuds IoT :** Chaque nœud représente un objet connecté simulé.Ces nœuds ne communiquent pas directement, toute tentative de communication passe par une authentification préalable via le KM. Il dispose :
   - Un identifiant unique (GUID),
   - Un vecteur binaire d'identité (ex. [1, 0, 1]),
   - Une clé privée (ex. BX92Z5GHYTQ1),
   - Un état is_active (actif ou en veille),
   - Une capacité à envoyer/recevoir des messages chiffrés.
2. **Le Key Manager (KM) :** Une fois la clé partagée obtenue, le KM déclenche la dérivation cryptographique (SHA-256) pour produire une clé AES 256 bits.Le KM a les responsabilités suivantes :
   - Enregistrer les nœuds avec leur identifiant, vecteur et clé privée.
   - Vérifier l'authenticité des nœuds lors d'un échange.
   - Générer une clé partagée basée sur la similarité entre vecteurs binaires et un sous-ensemble de la clé privée du nœud initiateur.

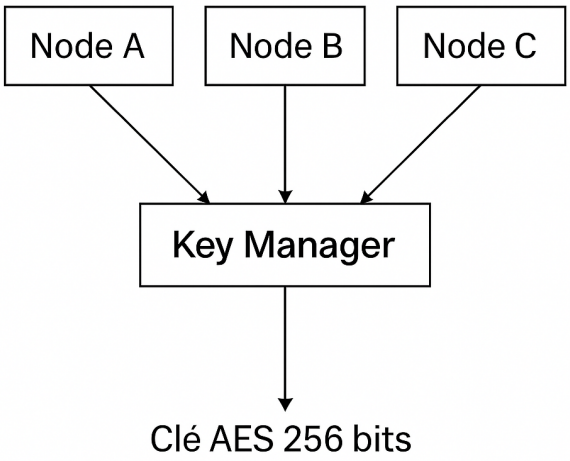


FIGURE 3.9 – Illustration du rôle central du Key Manager dans la génération des clés

3. **Le module de chiffrement :** Ce mécanisme assure la confidentialité, même pour des messages de faible longueur. IL repose sur :
   - l'algorithme AES (Advanced Encryption Standard),en mode CFB (Cipher Feedback Mode), adapté aux flux continus et messages courts.
   - Une clé dérivée unique par session.
   - Un vecteur d'initialisation [IV] aléatoire, généré à chaque chiffrement.

3.6.2 Schéma général de l'architecture

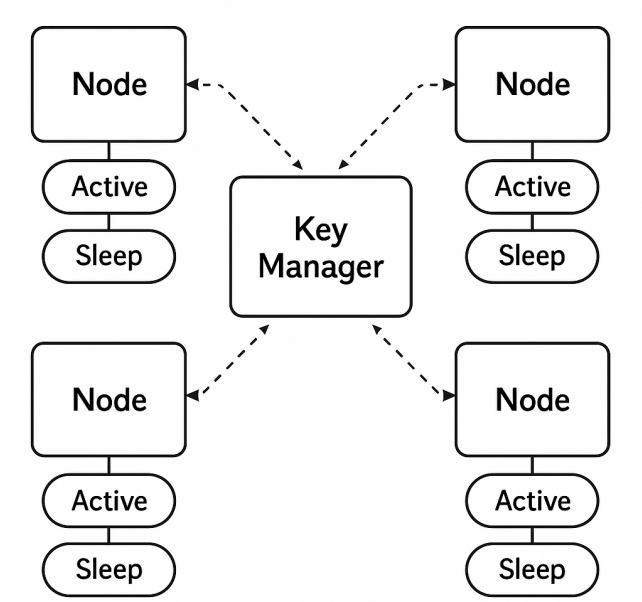


FIGURE 3.10 – Schéma d'architecture du système

3.6.3 Diagramme de séquence

Description du déroulement :

- Node A envoie une requête d'authentification → KM,
- KM vérifie, calcule une clé partagée, puis la dérive (SHA-256),
- Node A chiffre le message avec AES,
- Node B le reçoit et le déchiffre avec la même clé

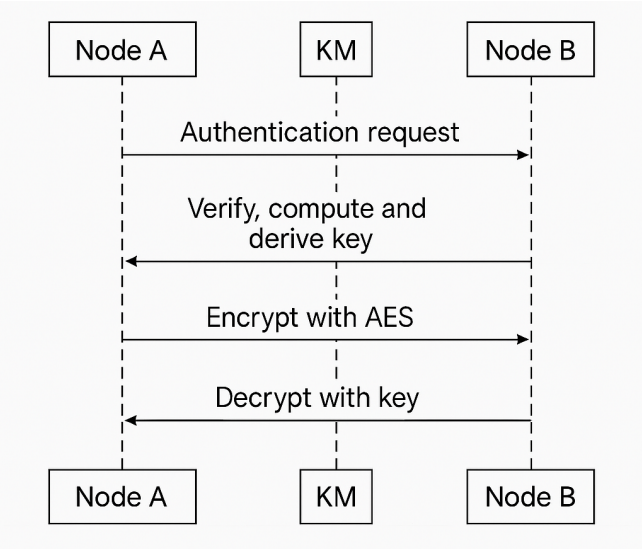


FIGURE 3.11 – Diagramme de séquence : communication sécurisée entre deux nœuds

3.6.4 Schéma interne du Key Manager (KM)

Le Key Manager (KM) constitue le cœur du protocole EVKMS simulé. Son rôle va au-delà d'un simple registre : il assure l'authentification, la dérivation des clés partagées, la validation des nœuds actifs, et le refus des communications illicites. Son fonctionnement interne peut être représenté par une série de modules coopérants.

Le schéma ci-dessous décrit les principales composantes internes du KM :

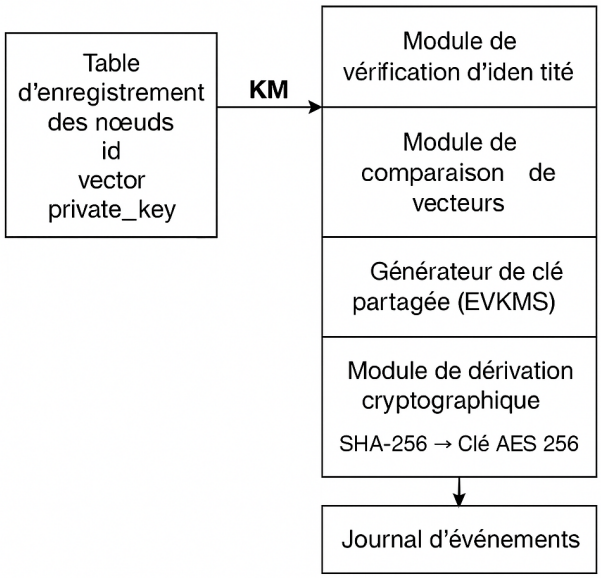


FIGURE 3.12 – Organisation interne du Key Manager

Description du schéma :

- **Table d'enregistrement des nœuds :**
  — Clé : id
  — Valeurs : vector, private_key
- **Module de vérification d'identité :**
  — Vérifie que les deux nœuds sont bien enregistrés
  — Rejette tout ID inconnu
- **Module de comparaison de vecteurs :**
  — Identifie les bits similaires entre deux vecteurs
- **Générateur de clé partagée (EVKMS) :**
  — Concatène bits similaires + préfixe de la clé privée
- **Module de dérivation cryptographique :**
  — Applique SHA-256 sur la clé partagée
  — Produit une clé AES 256 bits
- **Journal d'événements :**
  — Stocke les erreurs : tentatives de communication bloquées, rejets, etc...

3.7 Implémentation technique du protocole sécurisé A-EVKMS

3.7.1 Création et initialisation des nœuds

La première étape de l'implémentation consiste à générer un ensemble de nœuds. Chaque nœud est représenté par une structure de type dictionnaire Python contenant :

- Un identifiant unique généré aléatoirement (GUID).
- Un vecteur binaire aléatoire.
- Une clé privée propre au nœud.
- Un état (is_active) permettant de simuler l'économie d'énergie (actif/ veille).

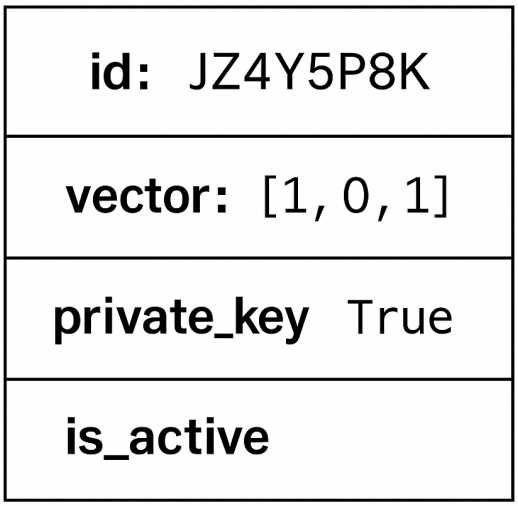


FIGURE 3.13 – Représentation des données d'un nœud (ID, vecteur, clé, état)

3.7.2 Enregistrement et gestion des identités via le KM

Le Key Manager (KM) joue un rôle central dans la gestion des identités. Lorsqu'un nœud est créé, il doit être enregistré dans le KM afin de pouvoir authentifier et sécuriser ses communications. L'enregistrement consiste à stocker pour chaque ID, son vecteur binaire et sa clé privée.

3.7.3 Dérivation d'une clé partagée

Lorsqu'un nœud A souhaite communiquer avec un nœud B, une clé partagée est dérivée à partir de la similarité entre leurs vecteurs binaires et d'un préfixe de la clé privée de A. Cette clé est ensuite hachée par SHA-256 pour produire une clé AES.

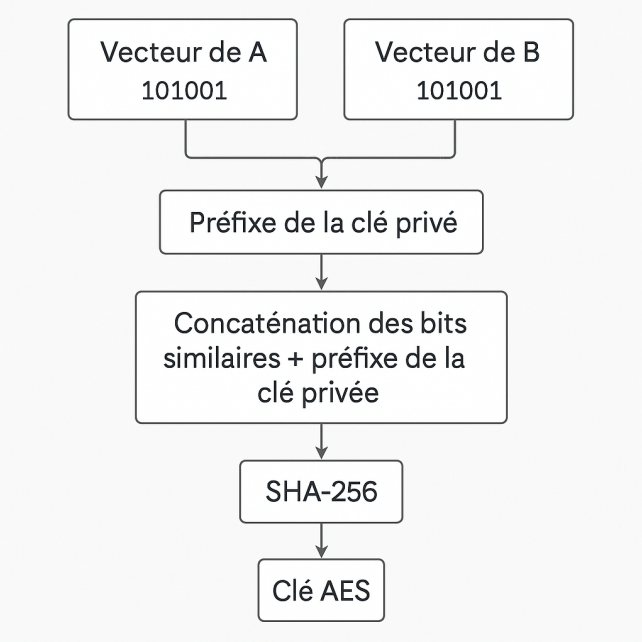


FIGURE 3.14 – Processus de dérivation de la clé (vecteurs + SHA)

3.7.4 Chiffrement et déchiffrement des messages

Le message est ensuite chiffré par AES en mode CFB, avec un IV aléatoire pour chaque session. Cela garantit un chiffrement unique même pour des messages identiques.

3.7.5 Simulation de communications entre nœuds

Une boucle permet de simuler des communications pair-à-pair entre tous les nœuds actifs. Les échanges sont authentifiés via le KM, chiffrés, puis déchiffrés côté récepteur [32].

3.7.6 Implémentation de l'état actif/veille des nœuds

Dans un souci de réalisme, chaque nœud dispose d'un attribut (is_active) permettant de simuler une mise en veille périodique, caractéristique des objets connectés à ressources limitées. Lors de chaque cycle de communication, une vérification est effectuée pour déterminer si le nœud est actif. Si ce n'est pas le cas, il est exclu du processus d'échange afin de refléter un comportement économe en énergie. Cette stratégie permet de mieux évaluer le protocole dans des conditions proches du terrain.

3.7.7 Simulation d'une attaque par interception (MITM)

Afin d'évaluer la robustesse du protocole A-EVKMS, une attaque de type "Man-In-The-Middle" a été simulée. Un nœud malveillant tente d'intercepter un message chiffré sans disposer de la clé de dérivation correcte. Lors du déchiffrement, le résultat est incohérent, voir inutilisable, confirmant ainsi que la confidentialité est assurée tant que la clé partagée reste inconnue de l'attaquant. Ce test illustre l'efficacité du chiffrement AES appliqué à une clé dérivée dynamiquement.

3.7.8 Rejet d'un nœud non enregistré

Une seconde simulation d'attaque consiste à introduire un nœud inconnu du Key Manager, ne disposant donc d'aucune entrée dans la base d'identités. Lorsqu'il tente d'engager une communication, l'authentification échoue immédiatement : son identifiant n'est pas reconnu et aucune clé de dérivation ne peut être générée. Cette mesure de contrôle empêche l'intégration d'éléments non autorisés dans le réseau, renforçant la sécurité globale du système.

3.7.9 Mesure des performances mémoire et temporelle

Pour évaluer l'efficacité de la solution A-EVKMS, des mesures ont été effectuées sur deux plans :

- **Temps d'exécution :** les durées nécessaires à la génération de la clé partagée, au chiffrement et au déchiffrement ont été mesurées à l'aide du module time.
- **Consommation mémoire :** l'empreinte mémoire des opérations a été mesurée à l'aide du module tracemalloc.

Ces résultats ont permis de comparer le protocole dans sa version sécurisée avec une version non chiffrée, mettant en évidence un surcoût raisonnable justifié par les gains en sécurité.

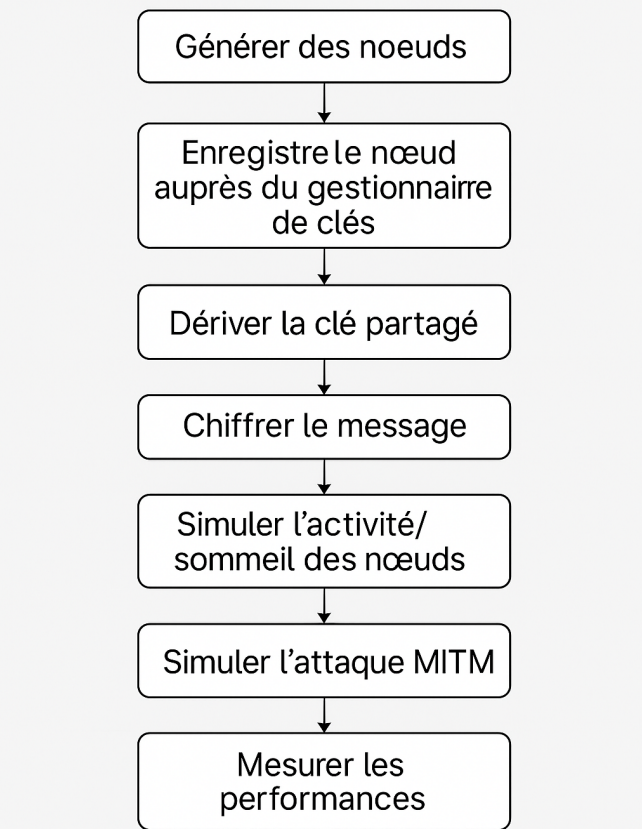


FIGURE 3.15 – Etapes Implémentation du protocole A-EVKMS

3.8 Évaluation de la solution A-EVKMS

3.8.1 Méthodologie d'évaluation

L'objectif de cette évaluation est de vérifier l'efficacité et la légèreté de notre protocole A-EVKMS dans un environnement contraint, en se focalisant sur la latence des opérations cryptographiques,la consommation mémoire et la résilience face aux attaques élémentaires.

La simulation a été menée sur un parc de 15 nœuds, avec deux nœuds mis en veille (is_active=False). Les opérations ont été mesurées dans un environnement Python, en local, à l'aide des bibliothèques time (pour les durées) et tracemalloc (pour l'usage mémoire).

3.8.2 Résultats sur le temps de traitement

Chaque communication simulée entre deux nœuds actifs comprend trois étapes mesurées :

- la dérivation de la clé AES (via SHA-256),
- le chiffrement du message en mode CFB,
- le déchiffrement par le destinataire.

La mesure détaillée a été extraite du log d'exécution Python. Voici quelques valeurs réelles observées dans notre simulation :

- Temps de chiffrement moyen : 1.00 ms
- Temps de déchiffrement moyen : 0.00 – 1.00 ms
- Temps total d'un échange : entre 1.00 ms et 1.20 ms, avec une seule valeur aberrante de 132.80 ms lors de la toute première communication.

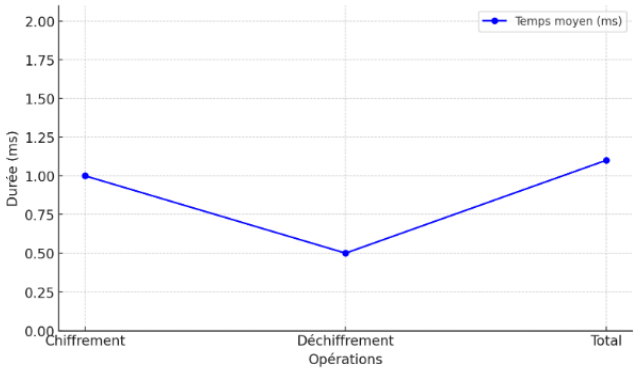


FIGURE 3.16 – Graphe linéaire des temps de traitement

3.8.3 Résultats sur la consommation mémoire

La mémoire consommée pour un échange crypté a été mesurée avec le module tracemalloc. La simulation d'un message chiffré/déchiffré a donné les résultats suivants :

- Mémoire utilisée (courant) : 2.61 KB
- Mémoire maximale atteinte : 3.72 KB

Ces résultats sont remarquablement faibles et montrent que l'implémentation est parfaitement adaptée aux systèmes embarqués disposant de très peu de mémoire, comme les microcontrôleurs 8-bit ou 16-bit.

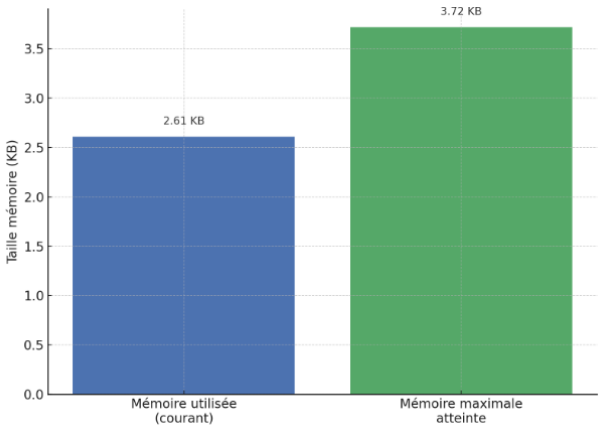


FIGURE 3.17 – Histogramme de la mémoire utilisée

3.8.4 Résultats sur les attaques simulées

- **Nœud non enregistré :** Un faux nœud, généré sans enregistrement préalable dans le Key Manager, a tenté d'initier une communication. Le protocole a détecté l'absence d'enregistrement dès la phase d'authentification et a bloqué l'échange.
- **Attaque Man-in-the-Middle (MITM) :** Un message intercepté par un attaquant a été chiffré avec une clé AES issue du protocole EVKMS. Le pirate, ne possédant pas la clé partagée dérivée, a tenté un déchiffrement avec une clé aléatoire.

Critère évalué	**Résultat observé**
Temps de chiffrement moyen	1.00 ms
Temps de déchiffrement	0.00 – 1.00 ms
Temps total d'échange	1.00 – 1.20 ms (hors anomalie)
Mémoire utilisée	2.61 KB
Mémoire max atteinte	3.72 KB
Attaque MITM	Échec du déchiffrement (✓)
Nœud non enregistré	Communication bloquée (✓)

TABLE 3.3 – Résumé des résultats de la simulation

3.8.5 Analyse critique des performances

Les résultats obtenus confirment la pertinence de l'architecture EVKMS dans un contexte IoT contraint : Les temps de traitement sont compatibles avec les exigences des systèmes en temps quasi-réel.

- La mémoire utilisée est largement inférieure aux capacités minimales des microcontrôleurs modernes.
- Le système est résilient face aux attaques classiques (MITM, intrus non enregistré), confirmant la solidité du mécanisme d'authentification basé sur les vecteurs partagés.

3.9 Comparaison entre la version de base et la version améliorée

Dans cette section, nous présentons une analyse comparative entre la première version basique de notre implémentation du protocole EVKMS et la version finale améliorée A-EVKMS. L'objectif de cette amélioration est de rendre le protocole plus réaliste, plus sécurisé, et plus proche des contraintes réelles rencontrées dans les systèmes IoT à faibles ressources.

3.9.1 Description de la version de base :

La version initiale du protocole EVKMS avait pour objectif principal de valider son principe fondamental dans un environnement restreint. Elle visait à simuler l'authentification et l'échange sécurisé entre deux nœuds à l'aide d'une clé dérivée, en intégrant la génération de cette clé à partir de vecteurs binaires et d'une clé privée partielle. De plus, la communication chiffrée entre les nœuds était testée via l'algorithme AES en mode CFB.

Sur le plan fonctionnel, cette première mouture inclut la création de cinq nœuds simulés, chacun avec son identifiant, un vecteur binaire et une clé privée. Ces nœuds étaient enregistrés dans un gestionnaire de clés simple basé sur un dictionnaire Python. Le protocole permettait alors de générer une clé partagée en identifiant les bits communs entre les vecteurs, enrichis d'un préfixe issu de la clé privée. La sécurité de la communication était assurée par le chiffrement et le déchiffrement du message via AES, et deux scénarios d'attaque étaient simulés : une attaque Man-in-the-Middle et une tentative de communication par un nœud non autorisé.

Cependant, malgré son bon fonctionnement, cette version reste limitée. Elle ne prenait pas en compte les états actifs ou veille des nœuds, un aspect pourtant crucial dans les systèmes IoT. Aucune métrique de performance, telle que le temps d'exécution, la consommation mémoire ou la latence, n'était mesurée. En outre, elle ne comportait aucun schéma technique illustrant les interactions, se limitait à un réseau de cinq nœuds seulement, et présentait un code peu lisible et non modulaire. Enfin, l'absence de journalisation ou de simulation de logs réseau représentait une autre lacune notable.

3.9.2 Détails des améliorations apportées :

Dans la version A-EVKMS a été étendu à un réseau dynamique de 15 nœuds, chacun généré avec un identifiant unique produit via un GUID personnalisé. Chaque nœud conserve un vecteur binaire de 3 bits, un choix aligné avec les contraintes mémoire typiques des microcontrôleurs utilisés dans l'IoT. Pour simuler les contraintes énergétiques courantes dans ces réseaux, un champ (is_active) a été introduit, permettant d'indiquer si un nœud est actif ou en veille. Deux nœuds ont ainsi été placés en veille simulée, empêchant toute tentative de communication avec eux. Par exemple, une tentative de communication entre B8FYZ1HQ et 5GJXZ6PQ a été automatiquement bloquée, ce qui reproduit fidèlement les interruptions fréquentes dans des environnements IoT.

Côté sécurité, les deux attaques simulées dans la version initiale ont été maintenues mais améliorées. Le rejet d'un nœud non inscrit est désormais explicite et journalisé, tandis que la simulation d'une attaque de type Man-in-the-Middle implique un déchiffrement forcé à l'aide d'une clé aléatoire, ce qui provoque systématiquement un échec. Un nœud malveillant ajouté dynamiquement est également reconnu comme non autorisé et immédiatement rejeté. Concernant les performances, de nouveaux modules ont été intégrés pour mesurer précisément l'impact du

protocole. Le temps de traitement est affiché avec un chiffrement à 1,00 ms, un déchiffrement à 0,00 ms, et un temps total de 1,20 ms. La mémoire consommée, mesurée via tracemalloc, atteint 2,61 KB, avec un pic à 3,72 KB.

Par ailleurs, cette version introduit plusieurs figures de visualisation destinées à clarifier le fonctionnement du protocole. On y trouve notamment un schéma de l'architecture générale avec le gestionnaire de clés (KM) au centre, un diagramme de séquence illustrant les échanges entre nœuds, des histogrammes montrant la latence et la mémoire utilisée, ainsi qu'un schéma détaillant l'organisation interne du KM et l'algorithme de dérivation de clé. Enfin, une attention particulière a été portée à la qualité du code. Celui-ci a été refondu pour adopter une structure modulaire, avec des fonctions distinctes pour chaque tâche, telles que l'enregistrement des nœuds, le chiffrement, ou la simulation d'attaques. Des commentaires explicites accompagnent chaque étape du processus. On a introduit des étapes dans la fonction principale afin de rendre l'exécution plus lisible et plus facile à maintenir.

3.9.3 Analyse comparative synthétique :

Critère	**Version de base**	**Version améliorée**
Nombre de nœuds	5	15
Gestion état veille	Non	Oui (champ `is_active`)
Attaques simulées	2 (MITM, nœud non inscrit)	2 améliorées + nœud malveillant bloqué
Mesure de performances	oui	Oui (temps + mémoire)
Modularité du code	Faible	Forte, fonctions séparées et commentées
Visualisation/figures	Aucune	Oui
Journalisation (logs)	Basique	Plus explicite, messages de contrôle
Réalisme IoT	Faible	Renforcé (veille, aléas, scalabilité)

Table 3.4 – Comparaison entre les deux versions du protocole

Table 3.5 – Comparaison des performances entre EVKMS et A-EVKMS

Critère évalué	**EVKMS (Base)**	**A-EVKMS (Amélioré)**
Nombre de nœuds simulés	5	15
État actif / veille simulé	Non	Oui (2 nœuds en veille)
Gestion d'identité	Basique	Améliorée (journalisation + rejet explicite)
Attaques simulées	MITM, nœud non enregistré	MITM, nœud non enregistré, nœud malveillant
Temps moyen de chiffrement	Non mesuré	1.00 ms
Temps de déchiffrement	Non mesuré	0.00 – 1.00 ms
Temps total d'un échange	Non mesuré	1.00 – 1.20 ms
Mémoire utilisée (courant)	Non mesurée	2.61 KB
Mémoire maximale atteinte	Non mesurée	3.72 KB
Visualisation / figures techniques	Aucune	Schémas, diagrammes, histogrammes
Modularité du code	Faible	Forte (fonctions séparées, commentaires, lisibilité)
Journalisation (logs)	Basique	Détails explicites sur le rejet ou l'erreur
Réalisme IoT simulé	Faible	Renforcé (veille, aléas, scalabilité)

La version améliorée du protocole A-EVKMS présente plusieurs avantages significatifs. Elle permet une implémentation plus fidèle des conditions réelles rencontrées dans les réseaux IoT contraints, en intégrant des mécanismes de gestion d'énergie et des interactions dynamiques entre les nœuds. La structure du code a été conçue de manière à être facilement adaptable à des plateformes embarquées courantes telles que l'ESP32 ou le STM32, ce qui facilite son déploiement dans des environnements matériels réels. En outre, la sécurité a été renforcée grâce à l'intégration de contrôles d'identité plus stricts, rendant le protocole plus résistant aux attaques classiques telles que les interceptions de type Man-in-the-Middle ou les tentatives d'injection de faux nœuds.

3.9.4 Perspectives possibles :

Quelques pistes peuvent encore être explorées :

- Intégration réelle dans un MQTT broker sécurisé (Mosquitto + TLS) [32].
- Portage du code Python vers C/C++ pour microcontrôleurs.
- Ajout d'un module de révocation dynamique des nœuds.
- Tests avec des vecteurs plus grands (8, 16, 32 bits).

3.10 Conclusion

Ce chapitre a permis de concrétiser la mise en œuvre du protocole EVKMS dans un environnement contraint, tenant compte des contraintes spécifiques aux objets connectés à faibles ressources. Grâce à une démarche progressive, un réseau de nœuds IoT a été modélisé afin d'établir des communications pair-à-pair sécurisées, reposant sur des vecteurs binaires pré-distribués, un gestionnaire de clés centralisé et un chiffrement symétrique par AES.

L'implémentation de base a validé les fondements du protocole de génération de clés partagées, fonctions de dérivation cryptographiques et échanges chiffrés. Une version étendue a ensuite été développée, appelée A-EVKMS avec plusieurs améliorations notables, augmentation du nombre de nœuds, gestion des états actif/veille, détection de comportements malveillants et simulation d'attaques comme l'interception (MITM) ou la tentative de connexion par un nœud non autorisé. L'intégration de mesures de performance (temps, mémoire) a permis d'évaluer l'efficacité du système.

Les résultats obtenus confirment que cette approche est adaptée aux systèmes embarqués simples, tout en assurant un bon niveau de sécurité. Bien que des améliorations soient possibles, notamment pour optimiser la consommation énergétique ou améliorer la scalabilité, le prototype prouve la pertinence d'A-EVKMS comme solution légère de gestion de clés pour l'IoT. Enfin, cette expérimentation ouvre la voie à des perspectives concrètes, telles que l'intégration avec des protocoles comme MQTT ou CoAP, ou encore le portage vers des plateformes matérielles réelles, comme des microcontrôleurs.

Conclusion générale

Ce travail de recherche nous a permis d'éxaminer un enjeu crucial et contemporain : la protection des communications dans les contextes IoT limités. Ces systèmes, généralement constitués de nœuds disposant de ressources restreintes, se sont imposés comme essentiels dans des secteurs clés tels que la domotique, la santé et l'industrie. Cependant, compte tenu de leurs contraintes matérielles, ils restent particulièrement exposés aux risques liés à la sécurité des données.

Dans un premier temps, nous avons posé les bases techniques et architecturales de l'Internet des Objets, en mettant en évidence les limites spécifiques des capteurs intelligents ainsi que les défis associés à la gestion sécurisée des communications. L'analyse des attaques courantes et des contraintes des systèmes a mis en lumière la nécessité de concevoir des mécanismes de sécurité à la fois légers, efficaces et adaptés à l'IoT. Nous avons ensuite étudié en détail le protocole EVKMS, basé sur la prédistribution d vecteurs secrets et la génération locale de clés. Grâce à son architecture simple et sa faible consommation en ressources, ce protocole s'est révélé particulièrement pertinent pour les environnements IoT. Ses caractéristiques en termes de confidentialité, d'intégrité et de résilience face aux attaques justifient son choix comme point de départ de notre implémentation.

Dans le cadre de ce travail, nous avons proposé une version améliorée du protocole, baptisée Adapted-EVKMS, dans laquelle plusieurs évolutions ont été intégrées : un mécanisme de gestion actif/veille des nœuds, une extension du réseau à quinze nœuds pour tester sa scalabilité, la prise en compte de scénarios d'attaque tels que l'introduction de nœuds non autorisés et les attaques de type Man-in-the-Middle, ainsi qu'une évaluation des performances en termes de temps de traitement et de consommation mémoire. Les résultats obtenus ont confirmé la légèreté et l'efficacité du protocole, avec une mémoire utilisée modérée, des délais de traitement très courts et une bonne résistance face aux tentatives d'intrusion.

Néanmoins, certaines limites subsistent. Il serait pertinent d'envisager à l'avenir l'intégration d'un mécanisme de révocation dynamique des nœuds, une meilleure gestion de la mobilité dans les réseaux, ainsi que le portage du protocole vers des microcontrôleurs réels. L'utilisation de protocoles de communication standardisés comme MQTT ou CoAP, combinée à cette solution de gestion de clés, représenterait également une avancée significative.

En somme, ce travail constitue une première étape vers la conception de solutions de sécurité réellement adaptées aux objets connectés contraints, conciliant simplicité, efficacité et robustesse dans un domaine en pleine évolution.

Bibliographie

Bibliographie

[1] Bettayeb, S., Messai, M.-L., & Hemam, S. (2023). *Efficient vector-based key management for constrained IoT environments*. Thèse de doctorat, Université de Béjaïa.

[2] Faux, S. (2017). La sécurité à l'ère des objets connectés : Comment s'y prendre ? Workshop *Sécurité des Objets Connectés*.

[3] Huitema, C. (1995). *IPv6 : le nouveau protocole Internet*. Prentice-Hall, Inc.

[4] SIG Bluetooth. (2010). *Spécification de base Bluetooth version 4.0*. Spécification du système Bluetooth, 1(7), 206.

[5] Msolli, A., Ajmi, N., Helali, A., Gassoumi, A., Maaref, H., & Mghaieth, R. (2022). Nouveau schéma de gestion de clés basé sur le hachage de pool pour WSN et IoT. *Journal of Communications and Information Technology*, 4(1).

[6] Azim, A., Rullier, A., Le Guennec, Y., Ros, L., & Maury, G. (2019). Modulation économe en énergie pour les communications en lumière visible dans le cadre de l'internet des objets. In *XXVIIème colloque GRETSI*. https://hal.archives-ouvertes.fr/hal-02304565/document

[7] SIG Bluetooth. (2010). *Spécification de base Bluetooth version 4.0*. Spécification du système Bluetooth, 1(7), 206.

[8] Rana, M., Mamun, Q., & Islam, R. (2023). Enhancing IoT security : An innovative key management system for lightweight block ciphers. *Sensors*, 23(18), 7678. https://doi.org/10.3390/s23187678

[9] Cryptography.io. (2024). Python Cryptography Module Documentation. https://cryptography.io/en/latest/

[10] Futura Sciences. Définition : Bluetooth. https://www.futura-sciences.com/tech/definitions/electronique-bluetooth-16527/

[11] Du, W., Deng, J., Han, Y. S., Varshney, P. K., Katz, J., & Khalili, A. (2005). A pairwise key predistribution scheme for wireless sensor networks. *ACM Transactions on Information and System Security*, 8(2), 228–258. https://doi.org/10.1145/1053283.1053289

[12] Gautam, A. K., & Kumar, R. (2021). A key management scheme using (p, q)-Lucas polynomials in wireless sensor network. *China Communications*, 18(3), 43–52. https://doi.org/10.23919/JCC.2021.03.007

[13] Liu, J., Wang, K., & Sun, H. (2023). Authenticated key agreement scheme for IoT networks exploiting lightweight linear algebraic computations. *Journal of Systems Architecture*.

[14] Zhao, N., Wang, S., & Yang, J. (2025). Research on Secure Authentication and Key Management of IoT Devices Based on Lightweight Protocols. In *Proceedings of ICAICT 2024*.

[15] AbuAlghanam, O., Qatawneh, M., Almobaideen, W., & Saadeh, M. (2022). Une nouvelle architecture hiérarchique et un nouveau protocole pour la distribution de clés dans les villes intelligentes. *Journal of Information Security and Applications*, 67, 103173.

[16] Zhang, Y., Deng, R. H., & Liu, X. (2014). Security and privacy in smart health : Efficient policy-hiding attribute-based access control. *IEEE Internet of Things Journal*, 1(5), 398–409. https://doi.org/10.1109/JIOT.2014.2343992

[17] Rana, M., Mamun, Q., & Islam, R. (2023). Enhancing IoT security... (réutilisé dans plusieurs sections).

[18] Zhang, Y., Xiang, Y., Huang, X., Chen, X., & Alelaiwi, A. (2018). Un protocole d'établissement de clés inter-couches basé sur une matrice pour les maisons intelligentes. *Sciences de l'information*, 429, 390–405.

[19] Du, W., Deng, J., Han, Y. S., Varshney, P. K., Katz, J., & Khalili, A. (2005). A pairwise key predistribution scheme for wireless sensor networks. *ACM TISSEC*, 8(2), 228–258.

[20] Mesmoudi, S., Benadda, B., & Mesmoudi, A. (2019). SKWN : Smart and dynamic key management scheme for wireless sensor networks. *International Journal of Communication Systems*, 32(7), e3930.

[21] Gautam, A. K., & Kumar, R. (2021). A key management scheme using (p, q)-Lucas polynomials in wireless sensor network. *China Communications*, 18(11), 210–228.

[22] Nafi, M., Bouzefrane, S., & Omar, M. (2020). Matrix-based key management scheme for IoT networks. *Ad Hoc Networks*, 97, 102003.

[23] Zeng, D., Guo, S., & Cheng, Z. (2016). The Web of Things : A Survey. *Journal of Communications and Networks*, 18(2), 1–14. https://doi.org/10.1109/JCN.2016.7406273

[24] Harb, H., & Missaoui, M. M. (2020). Lightweight mutual authentication protocol for IoT using physical unclonable functions. *Computers & Security*, 92, 101748. https://doi.org/10.1016/j.cose.2020.101748

[25] Housley, R. (2015). AES Encryption Performance on Embedded Systems. *IETF Journal*, 11(2), 22–27.

[26] Cryptography.io. (2024). Python Cryptography Module Documentation. https://cryptography.io/en/latest/

[27] Python Software Foundation. (2024). *hashlib module documentation*. https://docs.python.org/3/library/hashlib.html

[28] Python Software Foundation. (2024). *tracemalloc module documentation*. https://docs.python.org/3/library/tracemalloc.html

[29] Kaliski, B. S., et al. (2019). On the Security of AES : Attacks and Defenses in IoT Systems. *IEEE Access*.

[30] Python Software Foundation. (2024). *Documentation officielle Python*. https://docs.python.org/3/

[31] Mosquitto. (2023). *Mosquitto MQTT Broker – Documentation TLS*. https://mosquitto.org/man/mosquitto-tls-7.html

[32] Auteur. (2025). *Implémentation d'un protocole sécurisé EVKMS pour objets connectés contraints, simulation Python et rédaction de mémoire*.

[33] AbuAlghanam, O., Qatawneh, M., Almobaideen, W., & Saadeh, M. (2022). A new hierarchical architecture and protocol for key distribution in the context of IoT-based

smart cities. *Journal of Information Security and Applications*. https://doi.org/10.1016/j.jisa.2022.103240