



# Lightweight key management solutions for heterogeneous IoT

Mohamed Ali Kandi

## ► To cite this version:

Mohamed Ali Kandi. Lightweight key management solutions for heterogeneous IoT. Cryptography and Security [cs.CR]. Université de Technologie de Compiègne, 2020. English. NNT : 2020COMP2575 . tel-03214800

**HAL Id: tel-03214800**

**<https://theses.hal.science/tel-03214800>**

Submitted on 2 May 2021

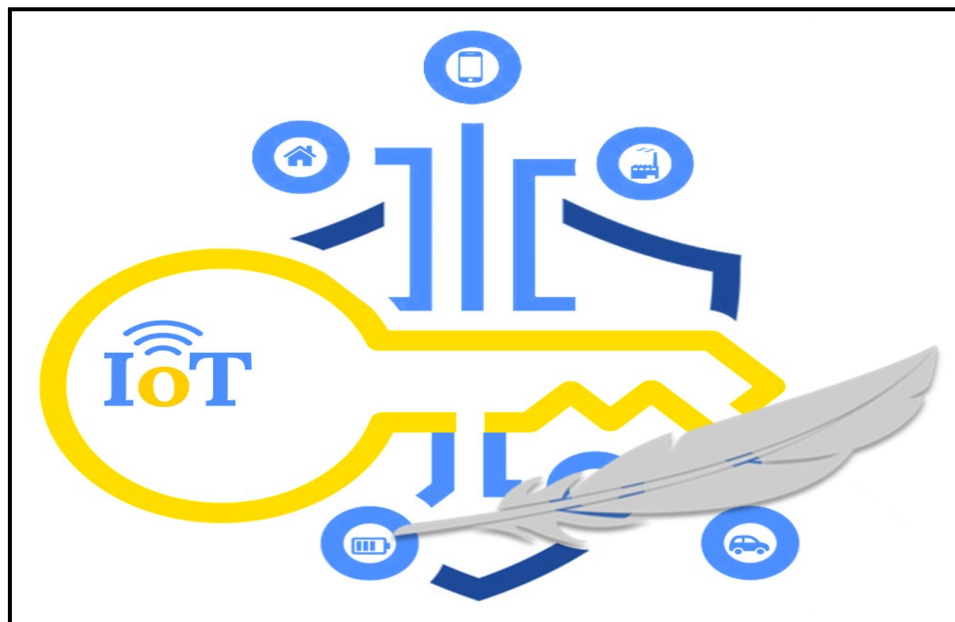
**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Par **Mohamed Ali KANDI**

*Lightweight key management solutions for  
heterogeneous IoT*

Thèse présentée  
pour l'obtention du grade  
de Docteur de l'UTC



Soutenue le 14 décembre 2020

**Spécialité :** Informatique et Sciences et Technologies de  
l'Information et des Systèmes : Unité de recherche Heudyasic  
(UMR-7253)

D2575

Thèse présentée pour l'obtention du grade de Docteur

## UNIVERSITÉ DE TECHNOLOGIE DE COMPIÈGNE

Spécialité : Informatique et Sciences et Technologies de l'Information et des Systèmes

---

# Lightweight Key Management Solutions for Heterogeneous IoT

14/12/2020

---

Par Mohamed Ali KANDI

### Composition du jury:

Abdelamadjid BOUABDALLAH	Professeur des universités, Université de Technologie de Compiègne	Directeur de Thèse
Karima BOUDAUD	Maître de conférences, Uni- versité de Nice	Examinatrice
Bernard COUSIN	Professeur des universités, Université de Rennes	Rapporteur
Romain LABORDE	Maître de conférences HDR, Université de Toulouse	Rapporteur
Hicham LAKHLEF	Maître de conférences, Uni- versité de Technologie de Compiègne	Co-Directeur de Thèse
Dritan NACE	Professeur des universités, Université de Technologie de Compiègne	Examineur





---

# *List of Publications*

---

## **Journal article**

- **Mohamed Ali Kandi**, Hicham Lakhlef, Abdelmadjid Bouabdallah, and Yacine Challal. “A Versatile Key Management Protocol for Secure Group and Device-to-Device Communication in the Internet of Things”. In: Journal of Network and Computer Applications 150 (2020).

## **International conferences**

- **Mohamed Ali Kandi**, Hicham Lakhlef, Abdelmadjid Bouabdallah, and Yacine Challal. “An Efficient Multi-Group Key Management Protocol for Internet of Things”. In: 26th IEEE International Conference on Software, Telecommunications and Computer Networks (SoftCOM). Split, Croatia, Sep. 2018.
- **Mohamed Ali Kandi**, Hicham Lakhlef, Abdelmadjid Bouabdallah, and Yacine Challal. “An Efficient Multi-Group Key Management Protocol for Heterogeneous IoT Devices”. In: IEEE Wireless Communications and Networking Conference (WCNC). Marrakech, Morocco, Avr. 2019.
- **Mohamed Ali Kandi**, Hicham Lakhlef, Abdelmadjid Bouabdallah, and Yacine Challal. “A Key Management Protocol for Secure Device-to-Device Communication in the Internet of Things”. In: IEEE Global Communications Conference (Globecom). Waikoloa, USA, Dec. 2019.

## **International workshop**

- **Mohamed Ali Kandi**, Djamel Eddine Kouicem, Hicham Lakhlef, Abdelmadjid Bouabdallah, and Yacine Challal. “A Blockchain-based Key Management Protocol for Secure Device-to-Device Communication in the IoT”. In: proceedings of the 19th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)/ International Workshop on Cyberspace Security (IWCSS). Guangzhou, China, Dec. 2020.

## Submitted article

- **Mohamed Ali Kandi**, Djamel Eddine Kouicem, Messaoud Doudou, Hicham Lakhlef, Abdelmadjid Bouabdallah, and Yacine Challal. “A Decentralized Blockchain-based Key Management Protocol for Heterogeneous and Dynamic IoT Devices”. In: IEEE Transactions on Dependable and Secure Computing. Submitted.

---

# *Abstract*

---

The Internet of Things (IoT) is an emerging technology that has the potential to improve our daily lives in a number of ways. It consists of extending connectivity beyond standard devices (such as computers, tablets and smartphones) to all everyday objects. The IoT devices, also called smart objects, can collect data from their surroundings, collaborate to process them and then act on their environment. This increases their functionalities and allow them to offer various services for the benefit of society. However, many challenges are slowing down the development of the IoT. Securing communication between its devices is one of the hardest issue that prevents this technology from revealing its full potential.

Cryptography provides a set of mechanisms to secure data. For their proper functioning, these mechanisms require secret parameters called keys. The Key Management is a branch of cryptography that encompasses all operations involving the handling of these keys: generation, storage, distribution and replacement. Lightweight cryptography consists of extending the conventional mechanisms (including the Key Management) to the resource-limited devices. To be efficient in the IoT, the new mechanisms must offer a good compromise between security, performance and resource requirements. Lightweight Key Management is the essence of secure communication in the IoT and the core of our work.

In this thesis, we propose a novel lightweight Key Management protocol to secure communication between the heterogeneous and dynamic IoT devices. To design our solution, we consider three modes of communication: device-to-device, group and multi-group communication. While most of the related works focus only on one of these modes of communication, our solution efficiently secures all three of them. It also automatically balances the loads between the heterogeneous devices according to their capabilities. We then prove that this makes our protocol more suitable for the IoT as it is efficient and highly scalable. Furthermore, we propose a decentralization of our protocol based on the blockchain technology and smart contracts. We show that, by empowering multiple participants to manage the cryptographic keys, decentralization solves trust issues, lowers risk of system failure and improves security. We finally implement our solution on resource-constrained IoT motes that are based on the Contiki operating system. The objective is to experimentally evaluate the performance of our solution and to complete our theoretical analyses.

---

**Keywords:** Internet of things, Security, Lightweight cryptography, Key Management, Blockchain.

---

# Résumé

---

L'Internet des objets (IdO) est une technologie émergente ayant le potentiel d'améliorer notre quotidien de différentes façons. Elle consiste à étendre la connectivité au-delà des appareils standards (tels que les ordinateurs, les tablettes et les smartphones) à tous les objets du quotidien. Ces appareils, également appelés objets intelligents, peuvent alors collecter des données de leur entourage, collaborer pour les traiter puis agir sur leur environnement. Cela augmente leurs fonctionnalités et leur permet d'offrir divers services au profit de la société. Cela dit, de nombreux défis ralentissent le développement de l'IdO. La sécurisation des communications entre ces appareils est l'un des problèmes les plus difficiles qui empêche cette technologie de révéler tout son potentiel.

La cryptographie fournit un ensemble de mécanismes permettant de sécuriser les données. Pour leur bon fonctionnement, ces derniers ont besoin de paramètres secrets appelés clés. La gestion des clés est une branche de la cryptographie qui englobe toutes les opérations impliquant la manipulation de ces clés: génération, stockage, distribution et remplacement. Par ailleurs, la cryptographie légère consiste à étendre les mécanismes conventionnels (la gestion des clés comprise) aux appareils à ressources limitées. Afin d'être efficaces dans l'IdO, les nouveaux mécanismes doivent offrir un bon compromis entre sécurité, performance et consommation de ressources. La gestion légère des clés est donc l'essence de la communication sécurisée dans l'IdO et le cœur de notre travail.

Dans cette thèse, nous proposons un nouveau protocole léger de gestion des clés pour sécuriser la communication entre les appareils hétérogènes et dynamiques de l'IdO. Pour concevoir notre solution, nous considérons trois modes de communication: d'appareil à appareil, de groupe et de multi-groupes. Alors que la plupart des travaux connexes se concentrent uniquement sur l'un de ces modes de communication, notre solution sécurise efficacement les trois. Aussi, elle équilibre automatiquement les charges entre les appareils hétérogènes en fonction de leurs capacités. Nous prouvons alors que cela rend notre protocole plus adapté à l'IdO étant donné qu'il est efficace et hautement évolutif. De plus, nous proposons une décentralisation de notre protocole basée sur la technologie blockchain et les contrats intelligents. Ainsi, nous montrons qu'en permettant à plusieurs participants

de gérer les clés cryptographiques, la décentralisation résout les problèmes de confiance, réduit le risque de défaillance du système et améliore la sécurité. Nous implémentons enfin notre solution sur des plateformes IoT à ressources limitées qui sont basées sur le système d'exploitation Contiki. L'objectif est d'évaluer expérimentalement les performances de notre solution et de compléter nos analyses théoriques.

**Mots Clés :** Internet des Objets, Sécurité, Cryptographie légère, Gestion des clés, Blockchain.

---

# *Contents*

---

List of Publications	iii
Abstract	v
Résumé	vii
Contents	ix
List of figures	xvii
List of tables	xxi
List of algorithms	xxiii
<b>1 Introduction</b>	<b>1</b>
1.1 Motivations . . . . .	1
1.2 Research topic . . . . .	2
1.3 Our contributions . . . . .	3
1.4 Organization of the manuscript . . . . .	4
<b>2 General context</b>	<b>5</b>
2.1 Introduction . . . . .	6
2.2 Fundamentals of the Internet of Things . . . . .	7
2.2.1 IoT architecture . . . . .	7

---

2.2.2	IoT features . . . . .	8
2.2.3	IoT applications . . . . .	10
2.2.4	Top IoT challenges . . . . .	11
2.3	Fundamentals of Network Security . . . . .	12
2.3.1	Network security objectives . . . . .	12
2.3.2	Network security attacks . . . . .	13
2.3.3	Network security and cryptography . . . . .	14
2.3.3.1	Ciphers . . . . .	14
2.3.3.2	Hash functions . . . . .	15
2.3.3.3	Key derivation functions . . . . .	16
2.4	Blockchain . . . . .	17
2.4.1	Blockchain structure . . . . .	17
2.4.2	Blockchain architecture . . . . .	18
2.4.3	Blockchain features . . . . .	19
2.4.4	Blockchain consensus . . . . .	20
2.4.5	Smart contracts . . . . .	21
2.5	Conclusion . . . . .	22
<b>3</b>	<b>Key Management in the IoT: Classification and Challenges</b>	<b>23</b>
3.1	Introduction . . . . .	24
3.2	Fundamentals of Key Management . . . . .	25
3.2.1	Generation . . . . .	25
3.2.2	Storage . . . . .	26
3.2.3	Distribution . . . . .	26
3.2.4	Replacement . . . . .	26



---

3.3	Key Management and IoT . . . . .	27
3.3.1	Lightweight cryptography . . . . .	27
3.3.2	IoT requirements for the Key Management . . . . .	28
3.4	Key Management classification . . . . .	30
3.4.1	Classification criteria . . . . .	30
3.4.1.1	Key cryptography . . . . .	30
3.4.1.2	Key type . . . . .	31
3.4.1.3	Distribution method . . . . .	32
3.4.1.4	Load balancing . . . . .	33
3.4.2	Proposed classification . . . . .	34
3.5	Key Management challenges in the IoT . . . . .	37
3.6	Our contributions . . . . .	38
3.6.1	Notations . . . . .	39
3.6.2	Application and threat model . . . . .	40
3.7	Conclusion . . . . .	41
<b>4</b>	<b>Dynamic Key Management for Secure Device-to-Device Communica-</b>	
	<b>tion</b>	<b>43</b>
4.1	Introduction . . . . .	44
4.2	Related Works . . . . .	45
4.2.1	Deterministic schemes . . . . .	45
4.2.2	Pure probabilistic schemes . . . . .	45
4.2.3	Deployment knowledge based schemes . . . . .	46
4.3	Our solution . . . . .	47
4.3.1	Classification of cryptographic keys . . . . .	48

---

4.3.2	Hash functions . . . . .	49
4.3.2.1	Zero-level approach . . . . .	49
4.3.2.2	One-level approach . . . . .	49
4.3.2.3	Two-level approach . . . . .	50
4.3.3	Set management . . . . .	52
4.3.3.1	Assignment Algorithm . . . . .	52
4.3.3.2	Reorder Algorithm . . . . .	53
4.3.4	Node management . . . . .	54
4.3.4.1	Node joining . . . . .	54
4.3.4.2	Node leaving . . . . .	55
4.4	Security analysis . . . . .	57
4.4.1	Theoretical analysis . . . . .	57
4.4.1.1	Zero-level approach . . . . .	57
4.4.1.2	One-level approach . . . . .	58
4.4.1.3	Two-level approach . . . . .	59
4.4.2	Comparison . . . . .	60
4.5	Performance evaluation . . . . .	61
4.5.1	Theoretical analysis . . . . .	61
4.5.1.1	Overheads on the Key Manager . . . . .	61
4.5.1.2	Overheads on the nodes . . . . .	62
4.5.2	Comparison . . . . .	62
4.5.2.1	Scalability . . . . .	63
4.5.2.2	Connectivity . . . . .	63
4.5.2.3	Mobility . . . . .	64

---

4.5.2.4	Efficiency . . . . .	64
4.5.2.5	Flexibility . . . . .	65
4.6	Conclusion . . . . .	65
<b>5</b>	<b>Heterogeneous Key Management for Secure Group Communication</b>	<b>67</b>
5.1	Introduction . . . . .	68
5.2	Related Works . . . . .	69
5.2.1	Tree based schemes . . . . .	69
5.2.2	Combinatorial optimization based schemes . . . . .	70
5.2.3	Batch rekeying based schemes . . . . .	70
5.3	Our solution . . . . .	71
5.3.1	Classification of cryptographic keys . . . . .	72
5.3.2	Subgroup Management . . . . .	73
5.3.2.1	Capability Evaluation Function . . . . .	74
5.3.2.2	Heterogeneous subgrouping . . . . .	74
5.3.2.3	Assignment Algorithm . . . . .	76
5.3.2.4	Reorder Algorithm . . . . .	79
5.3.3	Node management . . . . .	80
5.4	Security analysis . . . . .	81
5.4.1	Backward secrecy . . . . .	81
5.4.2	Forward secrecy . . . . .	81
5.4.3	Collusion resistance . . . . .	82
5.5	Performance evaluation . . . . .	82
5.5.1	Theoretical analysis . . . . .	82
5.5.1.1	Overheads on the Key Manager . . . . .	82

---

5.5.1.2	Overheads on nodes . . . . .	83
5.5.2	Simulation . . . . .	84
5.5.3	Comparison . . . . .	86
5.5.3.1	Efficiency and scalability . . . . .	87
5.5.3.2	Heterogeneity . . . . .	88
5.6	Conclusion . . . . .	89
<b>6</b>	<b>Blockchain-Based Decentralized Key Management for secure Multi-group Communication</b>	<b>91</b>
6.1	Introduction . . . . .	92
6.2	Related Works . . . . .	93
6.2.1	Multi-Group Key Management schemes . . . . .	93
6.2.2	Blockchain solutions . . . . .	93
6.3	Our solution . . . . .	94
6.3.1	Layer 1: Key Management . . . . .	95
6.3.1.1	Group and service management . . . . .	95
6.3.1.2	Classification of cryptographic keys . . . . .	96
6.3.1.3	Node Management . . . . .	97
6.3.1.4	Subgroup management . . . . .	97
6.3.2	Layer 2: Blockchain Management . . . . .	98
6.3.2.1	Transaction management upon network change . . . . .	99
6.3.2.2	Consensus Algorithm . . . . .	100
6.3.2.3	Blockchain interest . . . . .	101
6.4	Security analysis . . . . .	103
6.4.1	Independence of services . . . . .	103

---

6.4.2	Resilience against node capture . . . . .	104
6.4.2.1	Theoretical analysis . . . . .	104
6.4.2.2	Comparison . . . . .	105
6.4.3	Resilience against BP capture . . . . .	105
6.4.3.1	Theoretical analysis . . . . .	105
6.4.3.2	Comparison . . . . .	106
6.5	Performance evaluation . . . . .	107
6.5.1	Overheads on the Key Manager . . . . .	107
6.5.2	Overheads on nodes . . . . .	108
6.6	Conclusion . . . . .	109
<b>7</b>	<b>Experimentation</b>	<b>111</b>
7.1	Introduction . . . . .	112
7.2	Software environment . . . . .	113
7.3	Operating system: Contiki . . . . .	114
7.3.1	Processes . . . . .	114
7.3.2	Events . . . . .	114
7.3.3	Network Stack . . . . .	115
7.3.4	PowerTrace . . . . .	115
7.3.5	Cooja . . . . .	116
7.4	Material resource . . . . .	116
7.5	Experimental platform . . . . .	119
7.5.1	Key Manager . . . . .	119
7.5.2	Nodes . . . . .	119
7.5.3	Intermediate notes . . . . .	120

---

7.6	Experimental results . . . . .	121
7.6.1	Response time of <i>BPs</i> . . . . .	121
7.6.2	Storage overhead on nodes . . . . .	123
7.6.3	Execution time on nodes . . . . .	124
7.6.4	Energy consumption by nodes . . . . .	125
7.7	Conclusion . . . . .	127
8	<b>Conclusion and future works</b>	<b>129</b>
	<b>Bibliography</b>	<b>133</b>

---

## *List of figures*

---

2.1	IoT architecture. . . . .	7
2.2	IoT features. . . . .	8
2.3	IoT communication technologies. . . . .	9
2.4	Top IoT challenges. . . . .	11
2.5	Network security objectives. . . . .	12
2.6	Network security attacks. . . . .	13
2.7	Cryptographic hash functions. . . . .	16
2.8	Blockchain. . . . .	17
2.9	Client-server vs peer-to-peer networks. . . . .	18
2.10	Normal contract vs smart contract. . . . .	21
3.1	Key generation. . . . .	25
3.2	Lightweight Key Management. . . . .	27
3.3	IoT communication modes. . . . .	28
3.4	Classification criteria. . . . .	30
3.5	Key cryptography. . . . .	31
3.6	Key Type. . . . .	32
3.7	Distribution method. . . . .	33
3.8	Load balancing. . . . .	33
3.9	IoT network (Key Manager and nodes). . . . .	39

3.10	Threat model and countermeasures . . . . .	40
4.1	Key Management approaches for secure device-to-device communication. . .	46
4.2	Our device-to-device Key Management. . . . .	47
4.3	Example of node distribution in a network $N$ . . . . .	48
4.4	One-level approach. . . . .	50
4.5	Two-level approach. . . . .	51
4.6	Node joining and leaving. . . . .	56
4.7	An example of communications a node can decrypt. . . . .	57
4.8	An example of communications the node 3 cannot decrypt. . . . .	59
4.9	Variation of the percentage of compromised links. . . . .	60
4.10	Variation of nodes' storage overhead according to $n$ . . . . .	63
4.11	Variation of the average path length according to the size of the key ring. . .	64
5.1	Tree based schemes. . . . .	69
5.2	Key Management approaches for secure group communication. . . . .	70
5.3	Example of a group partitioning. . . . .	71
5.4	Our group Key Management. . . . .	72
5.5	Example of a group partitioned into three subgroups. . . . .	76
5.6	Example of a subgroup splitting. . . . .	78
5.7	Effect of the number of nodes on the value of $p$ . . . . .	85
5.8	Effect of the maximum capability of nodes on the value of $p$ . . . . .	85
5.9	Effect of the percentage of merging on the value of $p$ . . . . .	86
5.10	Effect of the subgrouping type on the value of $p$ . . . . .	86
5.11	Efficiency and scalability. . . . .	87
5.12	Heterogeneity. . . . .	88



6.1	Architecture of our solution. . . . .	94
6.2	Network partitioning. . . . .	96
6.3	Example of a blockchain transaction. . . . .	98
6.4	Decentralized rekeying upon a network change using a blockchain. . . . .	100
6.5	Rejoin exchange. . . . .	102
6.6	Resilience against node capture. . . . .	105
6.7	Resilience against $BP$ capture. . . . .	106
7.1	Structure of a Contiki process. . . . .	114
7.2	Rime stack overview. . . . .	115
7.3	Network components. . . . .	119
7.4	Key Manager. . . . .	119
7.5	Network partitioning. . . . .	120
7.6	Experimental platform. . . . .	120
7.7	Effect of $r$ on response time. . . . .	121
7.8	Effect of $nst$ on response time. . . . .	122
7.9	Effect of $cp$ on response time. . . . .	122
7.10	Comparison of our consensus algorithm with Tendermint. . . . .	123



---

## *List of tables*

---

2.1	Types of blockchain architecture. . . . .	19
3.1	IoT requirements for the Key Management. . . . .	29
3.2	Classification notations . . . . .	34
3.3	Classification of existing solutions . . . . .	36
3.4	Summary of notations. . . . .	39
4.1	Classification of cryptographic keys. . . . .	48
5.1	Example of keys held by nodes. . . . .	72
7.1	IoT mote specifications. . . . .	118
7.2	Storage overhead on nodes (Keys stored in the RAM). . . . .	123
7.3	Storage overhead on nodes (Keys stored in the flash). . . . .	124
7.4	Execution time on nodes (Keys stored in the RAM). . . . .	125
7.5	Execution time on nodes (Keys stored in the flash). . . . .	125
7.6	Energy consumption by nodes (Keys stored in the RAM). . . . .	125
7.7	Energy consumption by nodes (Keys stored in the flash). . . . .	126
7.8	Battery life. . . . .	126



---

## *List of algorithms*

---

1	Homogeneous Assignment Algorithm . . . . .	53
2	Homogeneous Reorder Algorithm . . . . .	54
3	Heterogeneous Assignment Algorithm . . . . .	77
4	Heterogeneous Reorder Algorithm . . . . .	80
5	Consensus Algorithm . . . . .	100



# *Introduction*

---

We are about to wake up to a new world. A world where, not only computers, but everything will be contacted to the Internet. From our clothing and appliances to our vehicles and buildings. A world where everyday objects will be so smart that they will be able to automatically interact with their environment and communicate with each other. They will be able to collect data from their surrounding, collaborate to process them and then act on their environment. This increases their functionalities and allow them to offer various services for the benefit of society, which until then were not able to provide. Known as the Internet of Things (or the IoT), this emerging technology promises to improve our daily lives in a number of ways. Smart homes, for example, involve using smart devices to ensure comfort, convenience and energy efficiency to the homeowners. Autonomous vehicles are able to automatically exchange data to maintain traffic flow and avoid crashes. However, many challenges are slowing down the development of the Internet of Things. Securing communication between its devices is one of the hardest issue that prevents this technology from revealing its full potential.

## **1.1 Motivations**

Network security consists of designing mechanisms to protect the data in motion from malicious attacks. In fact, securing communication is an old issue that existed long before the Internet of Things. It is as old as the computer networks themselves. Since then, there have always been people who target computer information systems, with or without malicious intent. Security becomes more and more complex as the motivations and capabilities of threat actors continue to evolve. The Internet of Things, despite its many benefits, has only made this task even more challenging. Today, securing communication is the primary concern of the Internet of Things developers and one of the main challenges that are slowing down the expansion of the technology [51]. This is mostly due to the three following reasons:

**Increasing number of devices:** The number of devices connected to the Internet is constantly increasing since its appearance. If all everyday objects will also be connected, this number will explode in the years to come. According to the Cisco Annual Internet Report [35], around *30* billion devices will be connected to the Internet by 2023 (up from *18.4* billion in 2018). This growing number of connected objects will necessarily generate a huge amount of data, which makes their protection much more challenging.

**Wireless and mobile devices:** Nowadays, mobile devices are evolving at lightning speed. This is largely due to the Internet of Things, since its very nature favors mobile to desktop devices. According to the Cisco Annual Internet Report [35], nearly *45%* of the devices that will be connected to the Internet by 2023 will be mobile (up from *30%* in 2018). On the one hand, these mobile devices necessarily require wireless communication. On the other, they must be physically small and thereby have limited energy supply. The problem is that securing communication becomes more difficult when it is wireless, especially if the energy resources are limited.

**Heterogeneous devices:** In addition to energy resources, the storage and computing capabilities are very varied in the Internet of things. They can also be very limited for some of its objects. Indeed, the Internet of Things devices include sophisticated servers, computers, smartphones as well as resource-constrained sensors. Balancing the level of security and performance between this variety of devices is very challenging.

## 1.2 Research topic

Cryptography is one of the strongest tool used to ensure security services. It includes a set of algorithms and mechanisms that are used to secure sensitive and classified data. Encryption, for example, can be applied to a block of data (plaintext) to convert it into a secret code (ciphertext) that hides the true meaning of the information. Cryptographic keys are the secret parameters that make the algorithms secure data. Like a physical key, a cryptographic one guarantees that only the entities owning it can open the door that protect the data. In encryption, for example, only people who know the right key should be able to retrieve the plaintext from a ciphertext. Therefore, for cryptography to be effective, it is important that the system that manages these keys is well designed. The Key Management is a branch of cryptography that encompasses all operations involving the handling of cryptographic keys: generation, storage, distribution and replacement [16].



---

The problem with cryptography, in general, is that it requires a significant amount of resources to be effective. Its algorithms can hardly be implemented in the constrained devices of the Internet of Things. Recently, a new concept, called lightweight cryptography, has appeared. It consists of expanding cryptographic mechanisms to the resource-limited devices, by designing new lightweight algorithms. These new mechanisms should be able to provide a good compromise between security, performance and resource requirements (hardware cost) [18]. The international standardization of lightweight cryptography is currently underway [96]. For lightweight cryptography to be efficient, it is essential that the Key Management can also operate over resource-limited devices. Lightweight Key Management is therefore the essence of secure communication in the Internet of Things and the core of our work.

### 1.3 Our contributions

In this thesis, we propose a novel lightweight Key Management protocol to secure communication between the heterogeneous and dynamic devices of the Internet of Things. To achieve this, we start by identifying the expectations of the Internet of Things from the Key Management. On this basis, we classify the related works according to different criteria. The objective is to clearly present the remaining challenges and to propose new solutions to overcome them. We then show that our protocol is more suitable for the Internet of Things than the existing ones. We also implement it on resource constrained IoT motes that are based on the Contiki operating system. The aim is to experimentally evaluate the performance of our protocol and to complete the theoretical analyses.

To design our solution, we consider three modes of communication: device-to-device, group and multi-group communication. In device-to-device communication, a device communicates with a specific other device. In group communication, a device communicates with a set of other devices. It can be its direct neighbors or all the network members participating in the same service. As a device can participate in several services at the same time, it may need to communicate only with those participating in a particular service. This is referred to as multi-group communication. While most of the related works only focus on one of these communication modes, our solution secures all three of them. It then considers the needs of each of them and best meets its requirements. To achieve this, we proceed in stages and propose several major contributions. These contributions are summarized in the following points:

**Dynamic Key Management for secure device-to-device communication:** We introduce a Key Management protocol for secure device-to-device communication in dynamic networks. Compared to the existing schemes, our solution provides a good compromise between the Internet of Things requirements: resilience, connectivity, mobility, flexibility, efficiency and scalability. To achieve this balance, the network members are uniformly distributed into logical sets. A device shares then a distinct pairwise key with each member of its set and a unique pairwise set key with the members of each of the other sets.

**Heterogeneous Key Management for secure group communication:** We enhance our protocol so that it also secures group communication in heterogeneous networks. Our new solution ensures the forward and backward secrecy and resists to collusion attacks. Also, by balancing the loads between the heterogeneous devices according to their capabilities, our solution becomes more efficient and highly scalable.

**Blockchain-based decentralized Key Management for multi-group communication:** We complete our solution so that it deals with the multi-group communication. We make sure that it guarantees the secure coexistence of several services in a single network. We also propose a decentralization of our protocol based on the blockchain technology and smart contracts. Thus, the system continues to operate when an entity fails and the compromise of a participant does not jeopardize the whole network.

## 1.4 Organization of the manuscript

The remainder of the manuscript is organized as follows. In chapter 2, we present the general context of our work: the Internet of Things, network security and the blockchain technology. In chapter 3, we detail the Key Management and emphasize its importance in network security. We also propose a classification of the existing solutions and identify the challenges they face in the Internet of Things. In chapter 4, we introduce a dynamic Key Management protocol for secure device-to-device communication in the Internet of things. In chapter 5, we present a heterogeneous Key Management protocol for secure group communication in the Internet of Things. In chapter 6, we introduce a blockchain-based decentralized Key Management protocol for multi-group communication in the Internet of Things. In chapter 7, we propose an implementation of our solution considering the Contiki operating system and resource constrained IoT platforms. Finally, we conclude our work, in chapter 8, and present the main future work directions and open issues.

## *General context*

---

In this chapter, we present the general context of our work and define the main concepts necessary to understand the rest of the manuscript. The first notion we introduce is the Internet of Things (IoT). This is a network made up of a large number of everyday objects, which are able to automatically communicate to computer systems, people and each other. The aim is to provide various services for the benefit of society. One of the main challenges that are slowing down the development of the IoT is how to secure communication between its devices. This leads us to the second concept related to our work, which is network security. Network security consists of designing a set of mechanisms to protect the network against existing and emerging threats. Some security protocols require a third party to function properly. This causes several problems, namely the dependence of a single point of failure and the need to trust a central entity. The blockchain, which is the last concept we introduce in this chapter, is a solution to these issues. It is a decentralized and secure storage technology. Based on smart contracts, the blockchain can be used to securely replicate an application on several entities.

## 2.1 Introduction

The number of devices connected to Internet is constantly increasing since its appearance. Now that this number far exceeds that of people in the world, we are no longer talking about Internet but about Internet of Things (IoT). The IoT devices, commonly called smart objects, are everyday objects that are able to automatically communicate to computer systems, people and each other. To achieve this, many software and hardware technologies are emerging. The IoT gives rise to revolutionary applications such as health care, environment monitoring, smart homes and smart cities. The aim is to provide various services for the benefit of society. One of the main challenges facing the development of the IoT is how to secure communication between this huge number of smart objects [51].

Network security consists of designing a set of policies and mechanisms to protect the network against existing and emerging threats. Although it is a fairly old discipline, it remains an open issue and the subject of numerous research. It gets even more complex over time since the motivations and capacities of threat actors continue to evolve. Despite its benefits, the IoT has exacerbated this problem even more. This is due to the wireless nature of communications, the limited resources of its devices and many other reasons [141]. Some security protocols require a third party for proper functioning. This creates several problems, namely the dependence of a single point of failure and the need to trust a central entity. The blockchain, which is a decentralized and secure storage technology, is a solution to these issues.

The term blockchain first appeared in Nakamoto's Bitcoin paper describing a new decentralized cryptocurrency [93]. The technology started then to be used in various applications. The blockchain is composed of a chain of blocks, each storing a cryptographic hash of the previous one. To guarantee that each network member records the same transactions in the same order, consensus algorithms are used. They define a set of rules to achieve overall system reliability in the presence of a number of faulty participants. The blockchain technology also allows the implementation of smart contracts. These are computer programs that are defined beforehand and stored in the blockchain. They can be run, automatically and in a decentralized way, by the blockchain participants to execute the settlement of a contract between organizations, people or objects.

The remainder of this chapter is organized as follows. Section 2.2 presents the fundamentals of the Internet of Things. Section 2.3 introduces the fundamentals of network security. Section 2.4 presents the blockchain technology and smart contracts. Section 2.5 concludes the chapter.

## 2.2 Fundamentals of the Internet of Things

The Internet of Things (IoT) is an emerging technology that promises to make our daily lives more comfortable in many ways. It aims to extend connectivity beyond standard devices (such as computers, tablets and smartphones) to all everyday objects. These devices, commonly known as smart objects, can then automatically collect data from their environment, store it, process it and even exchange it with each other. This enhances their functionalities and makes them able to offer new services to society, which until then were not able to provide.

### 2.2.1 IoT architecture

The common IoT architecture is composed of three layers: devices, gateways and cloud (Figure 2.1).

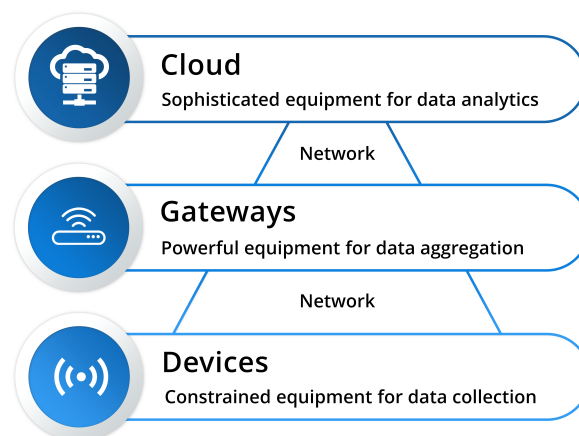


Figure 2.1 – IoT architecture.

**Devices:** The IoT devices, or smart objects, are everyday devices improved to increase their functionality (See section 2.2). A smart object involves an exchange between the physical and the digital words [72]. Let us take as example an alarm clock. If it is not smart, we can only set it to a specific time and it will alert us when the time comes. If the alarm clock is smart then it will be able to interact with its environment and communicate to other smart objects. It can, for example, communicate to the lighting so it gradually brightens our room in the morning. It can also communicate to the coffee maker and the heating, so that our place is warm and our coffee is ready and hot for us when we wake up. The IoT devices have the particularity of being heterogeneous (They are based on different technologies) and usually suffer from a lack of resources. For these reasons, IoT gateways are required [149].

**Gateways:** The IoT gateways are more powerful equipment than the IoT devices. They are mainly used as intermediate between other components. They can be used to translate communication between the heterogeneous devices if they need to exchange their data. They can also be used to store and process the data that the IoT devices can not handle for lack of capacity (whether in terms of storage or calculation). Finally, the IoT gateways can act as a middle layer between devices and cloud to protect the system from malicious attacks and unauthorized access [149].

**Cloud:** The IoT cloud is a sophisticated high performance network of servers optimized to perform high speed data processing and deliver accurate analyses. They are mainly used instead of gateways for two reasons: efficiency and remote access. The number of IoT devices being constantly increasing, the gateways may not be enough to efficiently manage the huge amount of data collected by all these devices. The IoT cloud offers then tools to collect, process, manage and store massive data in real time. Furthermore, the final user may need data collected by devices located far away. The IoT cloud can solve this problem by allowing remote access to the data collected by the devices [106].

### 2.2.2 IoT features

There are mainly four features that distinguish a smart object (IoT device) from a normal object: interaction with the environment (sensing and actuation), data management (processing and storage), communication and identification [72] (Figure 2.2).

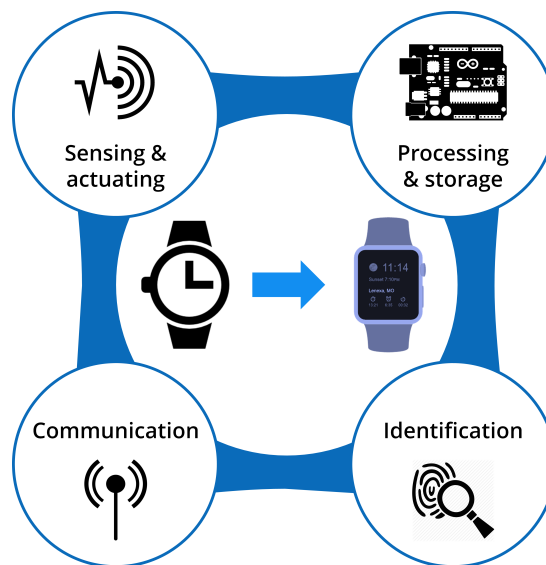


Figure 2.2 – IoT features.

**Interaction with the environment:** To be able to interact with its environment, i.e. collect data from it and act on it, a smart object must be equipped with sensors and actuators [52]. A sensor is an electronic device whose purpose is to detect events or changes in its environment. It is able to convert a physical, biological or chemical parameter (e.g. velocity, GPS coordinates, temperature, humidity, etc...) into an electrical signal that can be stored and processed by the device. On the opposite, an actuator transforms an electric signal into a physical parameter. Sensors and actuators act then as a bridge between the physical and the digital worlds.

**Data management:** Although most of calculation and storage are done on gateways and the cloud, smart objects are usually equipped with microcontrollers and non-volatile memories for local computing and storage [54]. The aim is to allow them to locally process the data they can handle and to autonomously make some decisions. This on-device intelligence will reduce network traffic and thereby energy consumption.

**Communication:** Smart objects must be able to communicate with each other and with the rest of IoT components. For this purpose, a number of different wireless communication technologies have been developed. They can be classified according to two axes: signal range and energy consumption [116] (Figure 2.3). Although a high-range communication provides a better connectivity, it is less secure. Indeed, it is easier to capture the traffic between two communicators. Low energy consumption technologies are usually more suitable for communications involving the IoT resource-constrained devices. However, they cannot achieve a high data rate.

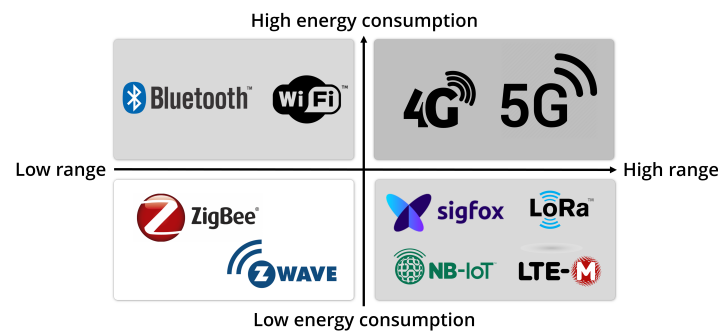


Figure 2.3 – IoT communication technologies.

**Identification:** To communicate, smart objects need to be able to identify one another. An identifier is like an electronic fingerprint that can take the form of a name, an address or a code. Several technologies are used to uniquely identify smart objects. Barcodes, for example, are optical representation of data, while RFID use Radio Frequency [66]. Therefore, unlike a barcode, an RFID tag does not need to be within the line of sight of the reader and may operate hundreds of meters from it.

### 2.2.3 IoT applications

The IoT makes possible the development of a huge number of applications, which have the potential to improve our lives. We can cite as examples: smart homes and cities, smart vehicles, smart health and smart environment.

**Smart homes and cities:** Smart objects distributed in houses and cities can make our lives more comfortable [55]. The room lighting can automatically change according to time of day. The heating may be adapted to our preferences and to the weather. Energy can be saved by automatically switching off the electrical equipments when not needed. Domestic incidents may be avoided with appropriate monitoring and alarm systems. Smart objects have also the ability to improve our cities. The standby lightning can switch on only when someone walks by. The air pollution and the CO<sub>2</sub> emissions may also be controlled to improve the environment [119].

**Smart vehicles:** Smart objects can be used by vehicles to sense their environment and autonomously navigate without human intervention. This will relieve travelers from driving and reduce accidents due to human errors [6]. Smart vehicles can also communicate with each other to control traffic, minimize congestion, find parking spots and avoid crashes.

**Smart health:** The global population is aging and the number of chronic diseases is increasing [100]. Basic healthcare will become out of reach to most people in the future. To resolve this problem, the IoT can be used to move the medical checks from hospitals to the patients' home. This will save a lot of time on both sides. Patients can then be provided with smart objects, which automatically collect health data (e.g. blood pressure, weight and blood sugar level) and share it with authorized people. These devices can also automatically alert hospitals in the case of a medical emergency like heart failure, diabetes, asthma attacks. Old people will be able to live without fear of not being able to call for help when they are alone.

**Smart environment:** Smart objects can be used to control and monitor the environment. They may detect a forest fire as soon as it starts. They then automatically deliver warning alarms so that firefighters can act before the fire spread. Smart objects can also be used to track endangered animals. Thus, they may live in their natural habitat, while being monitored all the time [95]. The IoT devices can also be deployed in ponds and tanks to monitor different parameters in real-time. The quality of the water can then be controlled to prevent some diseases that could affect people and animals.



### 2.2.4 Top IoT challenges

Although some of the IoT applications are currently available in our society, many challenges are slowing down their development. The main concerns of the IoT developers are [51]: data management, energy consumption, heterogeneity and especially security (Figure 2.4).





01	02	03	04
<b>Security</b> Securing communication between the wireless and resource constrained IoT devices is a hard issue. 	<b>Heterogeneity</b> Integrating the different materials, software and technologies in a single network cannot be done easily. 	<b>Energy</b> Powering the wireless IoT devices so that they remain operational for a long period of time is very difficult. 	<b>Data management</b> Managing the huge amount of data collected by billions of devices imposes significant challenges. 

Figure 2.4 – Top IoT challenges.

**Data Management:** In the future, billions of devices will be connected to the Internet. Managing the huge amount of heterogeneous data they can collect imposes significant challenges, especially when time, resource, and processing capabilities are limited.

**Energy consumption:** In most of the IoT applications, devices need to run in complete autonomy for several months or possibly years. However, these devices are usually wireless and limited by their small physical size. Consequently, they have restricted battery energy supply. It is therefore really difficult to power these devices to keep them operational for a long period of time.

**Heterogeneity:** Companies develop IoT protocols and materials independently of each other. This results in many different technologies that usually cannot integrate with one another. This lack of compatibility becomes a real issue. For example, as presented in the previous section, the IoT devices can use various communication technologies. These technologies being spread over different frequency, it is not possible to make all the IoT devices communicate directly.

**Security:** The IoT devices are vulnerable to cyber-attacks. Another task for engineers is then to secure them, especially when they collect sensitive data. However, having constrained computational resources, it is not possible to install any antivirus software on the IoT devices and the sophisticated security mechanisms are not efficient on them. Therefore, there is a need to design alternative security methods that are at least just as effective as the existing ones, while requiring fewer resources.

## 2.3 Fundamentals of Network Security

Security is the primary concern of the IoT developers and one of the main challenges that are slowing down the expansion of this technology [51]. Even before the appearance of the IoT, securing sensitive and classified data was considered as a major issue. It is becoming more and more complex since the motivations and capabilities of threat actors continue to evolve. The data to be protected can be either in rest (sitting on storage media) or in motion (moving across the network). Network security consists of designing mechanisms to protect the data in motion from attacks and to fulfil a certain number of objectives.

### 2.3.1 Network security objectives

Network security objectives usually involve three basic concepts, commonly referred to as the CIA triad [88]: confidentiality, integrity and availability (Figure 2.5).

**Confidentiality:** Confidentiality means that only the authorized entities must be able to view sensitive and classified data. Any other party must not be able to understand these data even if it can access them.

**Integrity:** Integrity means that only the authorized entities must be able to modify sensitive and classified data. An unauthorized modification on them must be detected.

**Availability:** Availability means that the data and system must be accessible to the authorized entities all the time. It must be available even in case of attempted attack or equipment failure.

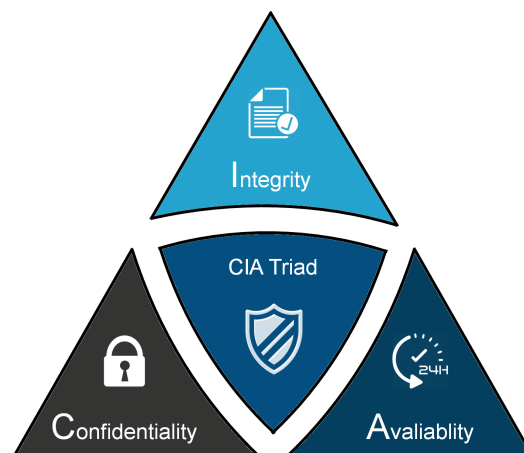


Figure 2.5 – Network security objectives.

### 2.3.2 Network security attacks

There are a number of threats that jeopardize the network security. Depending on the above-mentioned objectives, the network attacks can be classified into three categories [131]: passive, active and denial of service attacks.

**Passive attacks:** Passive attacks compromise confidentiality. They are usually based on eavesdropping as the attacker aims to access the data illegally without altering it (Figure 2.6a). Packet capturing and port scanning are examples of such attacks. In a packet capturing attack, the attacker captures the data packets crossing the network. He will then be able to read sensitive data like passwords or card numbers. In a port scanning attack, the attacker starts by searching for the TCP or UDP ports that are open on the target system. After that, he tries to discover the services running on these ports and the vulnerabilities in the software used.

**Active attacks:** Active attacks compromise integrity. They are usually considered as more dangerous attacks since the attacker does not just access the data but also modifies it (Figure 2.6b). He can then delete its content or inject new one. Man-in-the-middle attack is an example of such attacks. The attacker puts himself between two communicators to manipulate the packets exchanged between them.

**Denial of service attacks:** Denial of service (Dos) attacks aim to prevent a system from functioning properly and therefore compromise availability (Figure 2.6c). These attacks try to overwhelm the system by monopolizing or exhausting its resources. DoS attacks can be more severe if they are executed in a distributed manner.

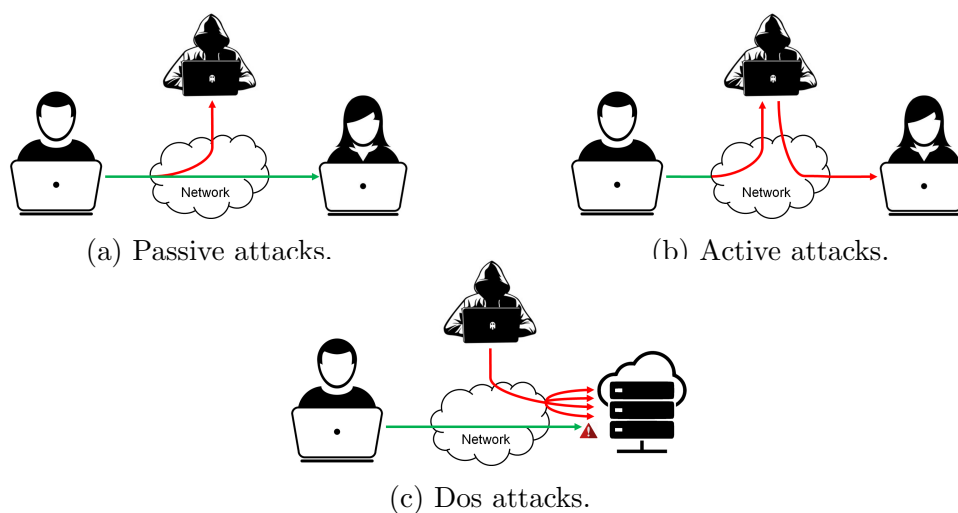


Figure 2.6 – Network security attacks.

### 2.3.3 Network security and cryptography

Cryptography is one of the strongest tool used to provide security services. It is considered both as an art of secret writing and a science used to protect sensitive and classified data. In addition to fulfilling two main security objectives (data confidentiality and integrity), cryptography ensures message authentication and non-repudiation [102]. In other words, it guarantees the authenticity of the author of a message and provides the proof that the announced author is the real one. Cryptography relies on two basic components: a key and an algorithm (or methodology). Keys are secret parameters that are usually known only by authorised entities. Algorithms are generally known to everyone. They are procedures that take as input data and keys and produce an output. There are mainly three kinds of such algorithms: ciphers, hash functions and key derivation functions.

#### 2.3.3.1 Ciphers

Ciphers are mathematical algorithms that can be applied to a block of data (plaintext) to convert it into a secret code (ciphertext) that hides the true meaning of the information. Ciphers must be reversible as it must be possible to retrieve the plaintext from the ciphertext. Encryption is the name given to the operation of using a cipher to generate the ciphertext, while decryption is used to refer to the reverse operation that consists of retrieving the plaintext. Ciphers being generally known to everyone, they are based on secret cryptographic keys to ensure confidentiality. An attacker, who does not have the appropriate keys, should not be able to retrieve the plaintext from a ciphertext. Ciphers can be classified into two categories: symmetric and asymmetric.

**Symmetric algorithms:** Also known as secret key algorithms, they involve the use of the same key for encryption and decryption. Their effectiveness depends on the secrecy of the shared keys. As long as the sender and receiver are the only ones to know it, they can securely exchange messages. Common examples of symmetric algorithms include the following: DES [85], 3DES [59], AES [33], and Blowfish [117].

**Asymmetric algorithms:** Also known as public key algorithms, they use different keys for encryption and decryption. Each network member is associated with a pair of related keys: a public key which is disseminated widely and a private key known only by it. The effectiveness of these algorithms depends then on the difficulty of guessing the private key from the public one. Common examples of asymmetric algorithms include the following: RSA [107], ElGamal [44] and ECC [20].

---

### 2.3.3.2 Hash functions

Cryptographic hash functions are mathematical operations that convert a block of data of arbitrary length to a fixed-size numerical value. This value should appear random and is called message digest or simply hash value. A cryptographic hash function must satisfy the following requirements [9]:

- It must be efficiently computable, meaning that the calculation of the hash must be easy and quick;
- It must be collusion resistant, i.e. it must not give the same hash for different input data;
- It must be a one-way function, in other words it is impossible (at least difficult) to find the input data from the hash;
- It must be deterministic so that the same input always results in the same hash;
- It must ensure that a small change to the input modifies the hash so that it appears uncorrelated with the first one.

Cryptographic hash functions can be used to verify data integrity. The sender may apply such a function on a message and attach to it the resulting hash. The receiver can then run the same function on the received message and compare the obtained hash with the one he received with the message. The three most popular cryptographic hash functions are MD5 [108], SHA-1 [43] and SHA-2 [122]. Just like ciphers, these functions are usually known to everyone. Therefore, they need to be combined with secret cryptographic keys to be effective. An attacker, who does not know the key used for generating the hash, should not be able to generate the same hash. There are mainly two methods of using a cryptographic hash function along with cryptographic keys.

**HMAC:** Hashed Message Authentication Code (HMAC) protocol consists of combining a cryptographic hash function along with a shared key to calculate a code that is attached to the message sent [74]. The receiver can then use the same key and hash function on the message and compare the result to the received code (Figure 2.7a). Thus, in addition to verifying the data integrity, the HMAC makes it possible to authenticate the sender of the message. Indeed, an attacker, who does not know the secret key, can not alter the message without being noticed since he cannot recalculate the correct hash.

**Digital signature:** While HMAC protocol uses symmetric keys along with a cryptographic hash function to generate an authentic code, digital signature uses asymmetric encryption [73]. The author of a message signs his message by hashing it and then using his private key to encrypt the obtained result. The receiver can decrypt the signature using the public key of the sender, calculate the hash of the received message and compare the two results (Figure 2.7b). If there is a match, the receiver will be able to ensure the data integrity, authenticate the sender of the message and be sure of its identity (non-repudiation). Indeed, being the only one to know his private key, no one else can generate the same digital signature.

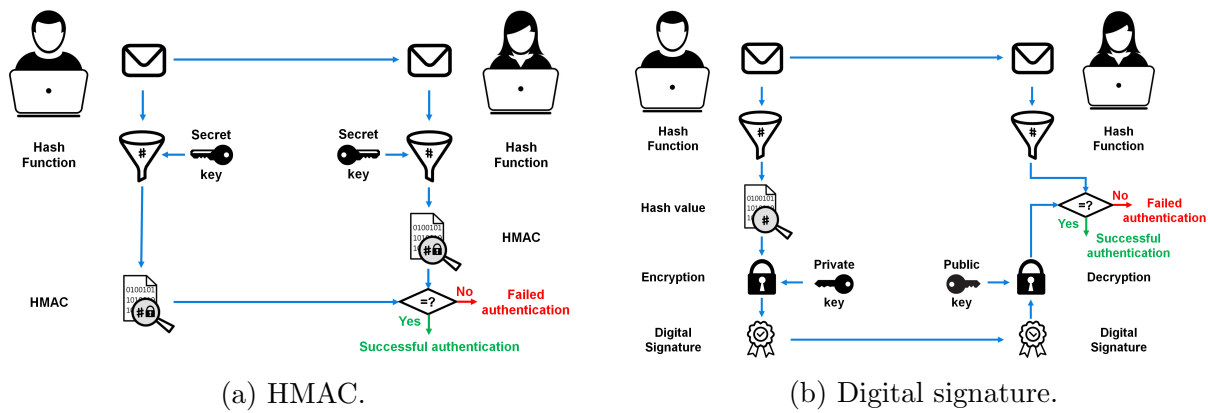


Figure 2.7 – Cryptographic hash functions.

### 2.3.3.3 Key derivation functions

Key derivation is a process by which a cryptographic key is derived from a password, a shared secret or an other key [16]. Key derivation functions must be deterministic so that the same input data always results in the same key. They are mainly used in the following three cases:

- They can be used when key expires to update them.
- They can be used when two or more participants collaborate to generate a shared key from several shared secrets.
- Passwords being usually weak, key derivation functions can be applied to them to generate longer and more random keys.

## 2.4 Blockchain

Given their complementary features, more and more applications that combines the IoT and the blockchain are emerging. The blockchain is a decentralized and secure storage technology. It is often confused with Bitcoin and cryptocurrencies in general. The term blockchain indeed first appeared in Nakamoto's Bitcoin paper describing a new decentralized cryptocurrency [93]. However, due to its many interesting characteristics, this technology is used today in various applications. We can cite as examples: food traceability, identity management and delivery of diplomas and certificates.

### 2.4.1 Blockchain structure

The term blockchain derives from the fact that it is composed of a chain of blocks, each storing a set of transactions and the cryptographic hash of the previous block [17] (Figure 2.8).

**Transaction:** It is the storage unit of the blockchain. The data stored in it depends on the application for which the blockchain is used. In Bitcoin, for example, a transaction contains data about the sender, the receiver and the amount of coins transferred.

**Block:** It is a data structure containing a set of transactions, their cryptographic hash and the hash of the previous block. Note that the first block does not store the hash of the previous one since there is none. In Bitcoin, for example, this is called the Genesis Block and is the first block ever created by Nakamoto himself.

**Chain:** It is a sequence of blocks in a specific order. The key behind blockchain's security is the fact that its blocks are linked by their hashes. Any change in a block requires modifying the hash of all the following ones in the chain, in other words, overwriting all its content.

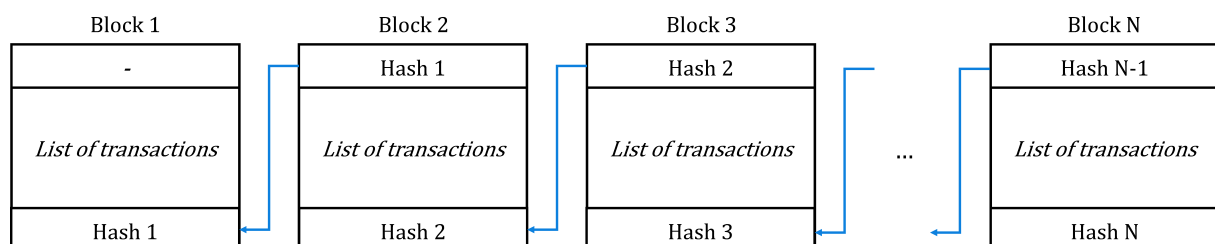


Figure 2.8 – Blockchain.

## 2.4.2 Blockchain architecture

Today, many applications use a client-server network (Figure 2.9a). This architecture is centralized since all data is stored in a single machine (the server) to facilitate its management. In a blockchain decentralized architecture, on the other hand, everyone stores and participates in the data management. The system is therefore controlled by every member of a peer-to-peer network [77] (Figure 2.9b). A blockchain perfectly illustrates the notion of democratized system. Since these collaborative parties do not necessarily trust each other, the blockchain technology offers mechanisms allowing them to reach common consensus. Thanks to that, the data cannot be altered without the agreement of the whole network, or at least most of them. The blockchain architectures can be classified into three categories [115]: public, consortium and private (Table 2.1).

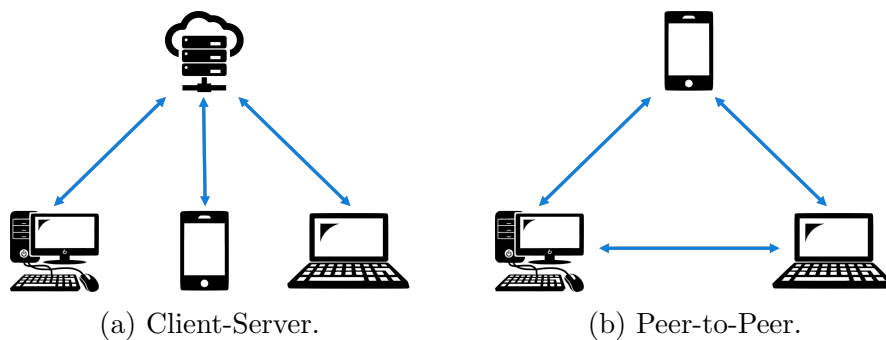


Figure 2.9 – Client-server vs peer-to-peer networks.

**Public blockchain:** In a public architecture, the blockchain is accessible and its data can be managed by anyone who wants to join. A large number of participants offers better immutability as it is not possible to modify the data after they have been stored in the blockchain. However, the data management requires a lot of resources. We can cite as examples of public blockchain: Bitcoin [93], Ethereum [139] and Litecoin [50].

**Consortium blockchain:** In a consortium architecture, the blockchain is accessible and its data can be managed only by authorized users from several organizations. With less participants, it becomes less difficult to alter the data. However, the blockchain management is more efficient. We can cite as examples of consortium blockchain: Quorum [10], Hyperledger [24] and Corda [22].

**Private blockchain:** In a private architecture, the blockchain is accessible and its data can be managed only by authorized users from a specific organization. With a limited number of participants, this architecture is the least secure, but the most efficient. We can cite as examples of private blockchain: Ripple [7], Tendermint [76] and Hyperledger [24].



Property	Public blockchain	Consortium blockchain	Private blockchain
Read permission	Public	Public or restricted	Public or restricted
Write permission	Public	Restricted	Restricted
Efficiency	Low	Medium	High
Immutability	High	Medium	Low
Example	Bitcoin	Hyperledger	Tendermint

Table 2.1 – Types of blockchain architecture.

### 2.4.3 Blockchain features

Regardless of the architecture adopted, the number of blockchain applications is constantly increasing. The prime reason behind this is that the blockchain is coming with myriads of features making it useful in various fields. We can cite as examples [84, 134]: decentralization, immutability, traceability, anonymity, transparency and security.

**Decentralization:** It consists of distributing computation and storage over multiple entities. This solves the single point of failure problem and makes it difficult for anyone to compromise the system.

**Immutability:** It means that the data stored in the blockchain are permanent and unalterable. They cannot be neither modified nor deleted. This brings more trust between parties and more data integrity.

**Traceability:** It is possible to track the origin of each transaction in the blockchain. This helps to prevent from tampering with the records and to quickly identify the source of a problem when it occurs.

**Anonymity:** If some users wish to hide their identity, the blockchain technology allows the use of generated addresses to keep their anonymity. They can then participate in the blockchain management without giving any personally identifying information.

**Transparency:** The blockchain content is viewable to everyone and its transactions are identically recorded in multiple locations. This is why the blockchain is considered hacking-resistant.

**Security:** The blockchain uses cryptography to ensure the confidentiality of the exchanged messages. It is more precisely based on digital signatures so that its participants can verify the integrity and authenticity of the transactions.

### 2.4.4 Blockchain consensus

The blockchain is a distributed database managed by several participants. A copy of its content is therefore stored in the memory of each of them. Before actually adding a new block to the blockchain, all (or at least the majority) of these participants must agree on it. They accomplish this agreement through consensus algorithms. These are mechanisms based on a set of rules to guarantee that each participant record the same blocks in the same order. The existing consensus algorithms can be classified into two categories [17]: leader-based mechanisms and byzantine fault tolerance-based algorithms.

**Leader-based mechanisms:** In this class of algorithms, the participants compete to elect the leader that will validate the next block. Proof of work (PoW) [93], as an example of such mechanisms, is mainly used in cryptocurrencies such as Bitcoin. Before adding a new block to the blockchain, each of its participants tries to solve a complex cryptographic puzzle. This process is referred to as mining and the individuals that participate in it as miners. The miners compete with each other to solve the puzzle and the first one to do it is rewarded with cryptocurrency. Once the puzzle is solved by a miner, it becomes the leader and forges the next block. All the other participants check then if the solution is correct and add the block to their copy of the blockchain. The verification must be an easy operation compared to the resolution. Solving the puzzle requires a lot of calculation and thereby wastes a huge amount of energy. Proof of Stake (PoS) [132] is an other leader-based mechanism, which is more energy-efficient. Indeed, the leader (also called validator) is chosen according to different criteria, without having to solve difficult problems. In cryptocurrencies, for example, the leader can be chosen according to its economic stake. Thus, a participant has a probability of being elected proportional to the amount of coins it possesses.

**Byzantine fault tolerance-based algorithms:** These algorithms are mainly based on communication. Using the Practical Byzantine Fault Tolerance Algorithm (PBFT) [25], for example, a participant (called proposer) is first elected in a round-robin fashion. The proposer broadcasts its block to all other participants, which becomes then validators. Each validator checks the received block, broadcasts a response (which can be positive or negative) and waits for the responses of the others. When a validator receives a number of positive responses above a certain threshold (two thirds of the number of participants in general), it broadcasts a commit and waits for the responses of the others. When a validator receives a number of commits above the threshold, it adds the block to the blockchain.

### 2.4.5 Smart contracts

Smart contracts are one of the most promising types of blockchain use. Concretely, they are autonomous and irrevocable computer programs usually stored in a blockchain. They can be automatically run by its participants to execute the settlement of a contract between organizations, people or objects [34]. Smart contracts are self-verifiable, self-executable and tamper proof. Although the idea of smart contracts was introduced in 1994, by Nick Szabo, it has not been put into practice until the appearance of the blockchain. It is precisely this technology that has eliminated the need for a trusted third-party. The advantages of smart contracts can be summarized in the following:

- Being stored in a blockchain, smart contracts take advantage of all its features to secure an agreement between two parties. We have indeed the guarantee that the terms of the contract cannot be modified.
- Unlike traditional contracts, automating the execution of smart contracts eliminates the risk of violation of its terms.
- Smart contracts reduce the intermediate costs: the preparation, the monitoring and the signing of a contract by notaries and lawyers.

Smart contracts are used, for example, in cryptocurrency to automatically exchange coins between users based on predefined conditions and without third-party involvement. They can also be used for travel insurance to automatically compensate passengers when their flights are late or cancelled. This operation is carried out without the need to fill out any form by clients or to process requests by companies. The sale of personal items is an other example of smart contracts use. The ownership of the item is automatically transferred to the buyer when the monetary is received by the seller. This can be done without the need of a third-party and without risk of fraud (Figure2.10).

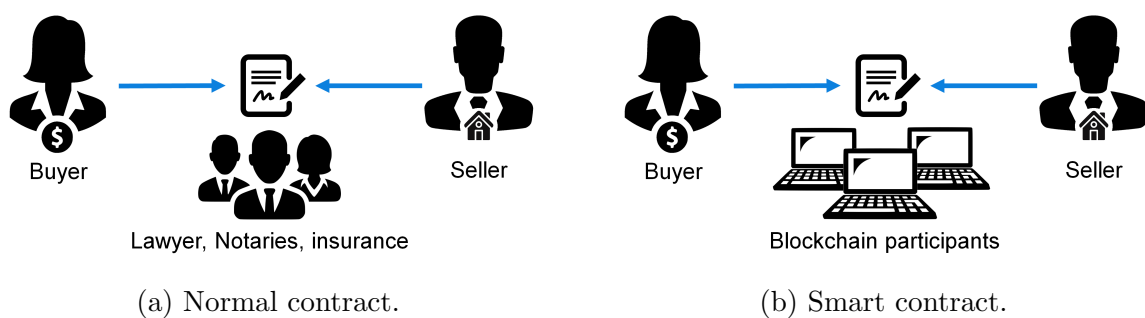


Figure 2.10 – Normal contract vs smart contract.

## 2.5 Conclusion

The purpose of this chapter was to present the general context of our work. We started by introducing the concept of the IoT, its architecture, features and applications. We also stated that the main challenge that is slowing down the development of this emerging technology is security. We then recalled the fundamentals of network security and the cryptographic mechanisms that are used to achieve its goals. We finally presented the blockchain and smart contracts, which are other emerging technologies that are closely linked to the two other concepts.

An important point to consider is that most of the cryptographic mechanisms, that we covered in this chapter, require keys for their proper functioning. The key to security is therefore the cryptographic keys. For this reason, the next chapter is entirely devoted to the concept of Key Management for secure communication between the IoT devices.

# *Key Management in the IoT: Classification and Challenges*

---

In this chapter, we present the objective of our research and the related works. We start by defining the notion of Key Management and its importance in network security. The role of such a system can be summarized in the generation, storage, distribution and revocation of cryptographic keys. Next, we present different criteria that can be used to classify the existing solutions. These criteria are related to the encryption technique used, the type of key, the distribution method and the load balancing. We then present some of the existing Key Management protocols and classify them according to the previously mentioned criteria. We also discuss each of this categories by citing its advantages and disadvantages. The aim is to identify the challenges that the related works encounter. Finally, we summarize our contributions to overcome these challenges.

## 3.1 Introduction

Cryptographic keys are secret parameters that are combined with cryptographic algorithms to secure communication. They are used to encrypt data, generate HMAC codes, sign messages and even derive other keys. Thus, only the entities that know the key must be able to reproduce or reverse the operation for which it was used, i.e. retrieve the plaintext from a ciphertext, verify the HMAC code, check the digital signature or generate the same key. Since the effectiveness of cryptographic algorithms depends on keys, it is important that the system that manages these keys is well designed. The main role of a Key Management system is therefore to establish secure links between the members of a network by providing them with cryptographic keys [146].

The Key Management system is first responsible of generating keys. It also stores them, and sometimes archives them, when necessary. It is in charge of distributing keys on the appropriate network members as well. Finally, it updates the keys when new members join the network and replace them when they get compromised. Although different Key Management protocols were proposed, each of them presents its own limitations and weaknesses. More importantly, our literature revue shows that none of the existing solutions meets all the IoT requirements in terms of security and performance. This leads us to present lightweight cryptography, which aims to use less resources than conventional cryptography in terms of storage, computing, communication and energy. The goal is to provide security solutions that can operate over resource-limited devices [23].

To properly categorize the existing solutions and to determine the problems they encounter, we classify them according to several criteria: the encryption technique used, the type of key, the distribution method and the load balancing. The combination of these different parameters gives rise to a number of protocol classes. We discuss the advantages and disadvantages of each of them. This classification helped us to achieve a comprehensive view of the exiting Key Management protocols. We were then able to clearly identify the challenges that these solutions face in the IoT. The Key Management being the purpose of our work, we propose novel contributions to overcome these challenges. Unlike most of the existing protocols, our solution is more IoT oriented and better meet its needs.

The remainder of this chapter is organized as follows. Section 3.2 presents the fundamentals of the Key Management. Section 3.3 introduces the IoT requirements for the Key Management. Section 3.4 presents a classification of the existing Key management solutions. Section 3.5 introduces the Key Management challenges in the IoT. Section 3.6 presents our contributions. Section 3.7 concludes the chapter.

## 3.2 Fundamentals of Key Management

The proper management of keys is essential to guarantee the effectiveness of cryptography. Indeed, if an adversary manages to discover the keys, he will be able to thwart all the cryptographic mechanisms. The Key Management is therefore the essence of secure communication and the core of our work. It includes all the operations involving the handling of cryptographic keys: generation, storage, distribution and replacement [16].

### 3.2.1 Generation

Keys can be generated by their owner(s) or by a trusted third authority. This is mainly done in three different ways: random generation, key derivation and key agreement [12]. Note that some Key Management protocols have specific methods for generating keys.

**Random generation:** Keys can be generated using a pseudo-random bit generator (RNG). This is a device or an algorithm that outputs numbers with properties close to that of sequences of random numbers (Figure 3.1a) [12].

**Key derivation:** Keys can be derived from a password or another key using a key derivation function (KDF). Passwords being usually weak, key derivation functions can be applied to them to generate longer and more random keys. Moreover, an expired or compromised key can be combined with an available one to drive a new key (Figure 3.1b) [14].

**Key agreement:** Keys can be generated with agreement between participants using a key agreement protocol (KAP) such as Diffie-helman[38]. The aim is to obtain the key by merging secret information from several entities (Figure 3.1c) [13]. The combination of these information can be done by concatenating or exclusive-oring them for example.

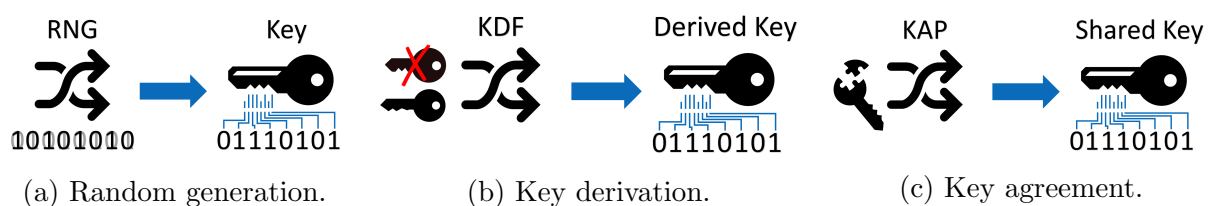


Figure 3.1 – Key generation.

### 3.2.2 Storage

The time span during which a key is authorized for use is called cryptoperiod [14]. During this phase, the key must be available to the appropriate devices. Keys can be stored within devices if they are being actively used. They may also be stored in more secure external storage mediums and be recalled when needed. In some cases, keys can even be archived at the end of their cryptoperiod, so that they can be recovered later [16]. Archived keys are usually encrypted to ensure their confidentiality.

### 3.2.3 Distribution

Distribution consists of moving a key from an entity that either owns or generates it to another that is intended to use it [12]. When secret key distribution is required, the keys must be transported manually or using secure channels. Manual transporting means physically moving a device or document containing the key. A secure channel is a path that uses cryptography to ensure confidentiality and integrity [14]. The distribution of public keys does not have to be secured, but should ensure to the receiver that the claimed owner is the actual owner. A trusted entity is usually required to vouch for the identity of users.

### 3.2.4 Replacement

Keys need to be replaced in two cases: expiration and revocation [14]. A key expires when its cryptoperiod is over. If a key gets compromised before the end of its cryptoperiod, it will be revoked and will no longer be operational. Key replacement can be done in two different ways: rekeying and key update.

**Rekeying:** The new key is independent of the old one [13]. As long as an adversary does not have access to the new key, he will not be able to calculate it even if he knows the old one. This method is more secure and well suited in case the old key is compromised. It requires nevertheless the distribution of the new key.

**Key update:** The new key is derived from the old one in a non-reversible way [16]. This method is more efficient as all entities that share the key can update it independently of each other. However, an adversary who knows the old key can calculate the new one. Thus, key update is usually used when a key expired. This method can still be applied on compromised keys, if it combines them with some secret parameters.



### 3.3 Key Management and IoT

Conventional cryptographic mechanisms, in general, and conventional Key Management, in particular, are effective in traditional Internet. Desktop and server environments have indeed enough resources in terms of storage, computing, communication and energy. However, these mechanisms cannot or can hardly be implemented in the IoT resource-constrained devices. Even when this implementation is possible, it is not efficient and does not scale well. New lightweight solutions, including Key Management, are therefore required to overcome many of the problems of conventional cryptography [23].

#### 3.3.1 Lightweight cryptography

The purpose of lightweight cryptography is to expand cryptographic mechanisms to resource-limited devices. It consists then of proposing new lightweight solutions or adapting the existing ones so that they become suitable for implementation on these devices. To achieve this, these solutions must provide the best compromise between security, performance and resource requirements (hardware cost) [18]. Figure 3.2 illustrates an example of the difficulty of satisfying these three contradictory criteria.

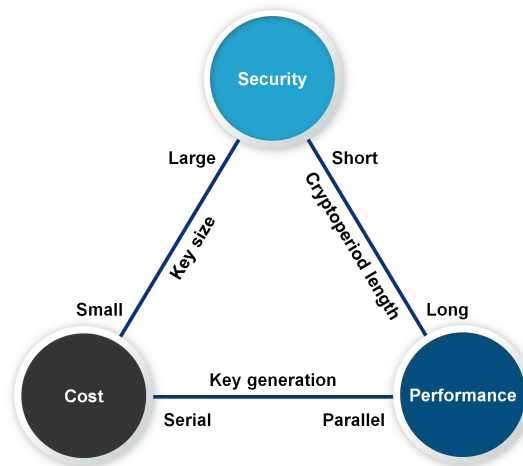


Figure 3.2 – Lightweight Key Management.

Lightweight cryptography is a relatively young scientific sub-field of cryptography whose international standardization is currently underway. The international standard ISO/IEC 29192 (Lightweight Cryptography) was established at ISO/IEC JTC 1/SC 27. The U.S. National Institute of Standards and Technology (NIST) launched the Lightweight Cryptography Project in 2013 and announced a public call for applications of lightweight cryptographies in 2017 [96].

### 3.3.2 IoT requirements for the Key Management

Although lightweight Key Management protocols have been proposed, most of them are intended for homogeneous and static wireless sensor networks. They rarely consider the heterogeneous and dynamic nature of the IoT and none of them meets all its requirements. Thus, we classify the IoT expectations for the Key Management according to the communication mode. The IoT devices may indeed communicate in three different ways: device-to-device, group and multi-group communication (Figure 3.3). Each of these communication modes has different requirements for the Key Management (Table 3.1).

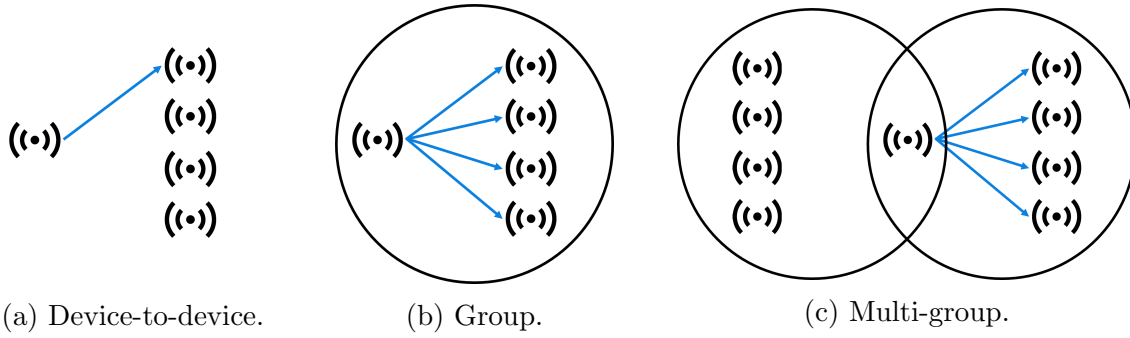


Figure 3.3 – IoT communication modes.

**Device-to-device communication:** A device sends/receives unicast messages to/from another specific device (Figure 3.3a) [101]. An example of device-to-device communication (also called node-to-node or pairwise communication) is the Vehicle-to-Vehicle communication. It is a technology that allows two specific vehicles to exchange information about their speed and position. Thus, they can avoid crashes, ease traffic congestion and improve the environment [28]. The device-to-device communication requires that the Key Management provides good resilience, connectivity, mobility, flexibility, efficiency and scalability (See Table 3.1).

**Group communication:** A device sends/receives broadcast messages to/from the members of the group to which it belongs (Figure 3.3b) [129]. This device must have previously joined the group legally. It can also voluntarily leave it afterwards or be evicted if it gets compromised. The group members usually participate in the same service and thereby have a common interest. An example of group communication is the Vehicle-to-Everything communication. It consists of allowing a vehicle to communicate with all the nearby devices (cars, bicycles, public lighting...etc.). The aim is to make the vehicle sense its environment and therefore take the right decision [28]. The group communication requires that the Key Management ensures backward and forward secrecy, collusion resistance, efficiency and scalability (See Table 3.1).

**Multi-group communication:** A device sends/receives multicast messages only to/from the members of one of the groups to which it belongs (Figure 3.3c) [62]. Unlike group communication, where a device is supposed to belong to a single group, this mode considers the possibility that devices can participate in multiple services at the same time. An example of multi-group communication is smart ambulances [113]. They can participate at the same time in the intelligent transportation and healthcare systems. The multi-group communication requires that the Key Management guarantees independence of services, efficiency and scalability (See Table 3.1).

Communication mode	Requirement	Description
Device-to-device	Resilience	The capturing of a device must have a minimal impact on the network security
	Connectivity	The probability of sharing keys between neighboring devices must be maximum. Otherwise, they must relay on intermediate devices to establish secure paths
	Mobility	Moving devices must share keys with their new neighbours
	Flexibility	Devices must be able to securely join or leave the network at any time
Group	Backward secrecy	New devices must not have access to the old keys. Thus, when a device joins the network, the keys must be replaced
	Forward secrecy	Old members must not have access to the future keys. Thus, when a device leaves the network, the keys must be replaced
	Collusion resistance	Unauthorized devices must not have access to the keys if they cooperate
Multi-group	Independence of services	The compromise of a service must have no effect on the others
All three modes	Efficiency	Since most of the IoT device suffer from a lack of resources, the use of these resources by the Key Management must be minimal
	Scalability	Given the tremendous number of IoT devices, increasing the network size must not degrade the Key Management performance

Table 3.1 – IoT requirements for the Key Management.

## 3.4 Key Management classification

After identifying the expectations of the IoT from the Key Management, we now analyze the existing solutions. To properly characterize these solutions, we start by classifying them according to different criteria. We then discuss each category and deduce the IoT requirements it meets and those it does not. The objective is to clearly identify the remaining challenges and to propose new solutions to overcome them.

### 3.4.1 Classification criteria

After conducting an extensive literature review, we have retained four essential criteria to classify the existent Key Management solutions: the key cryptography, the key type, the distribution method and the load balancing. The first two concern the keys themselves, while the other two relate to how they are managed. Two levels of classification can therefore be observed: Key level and Management level (Figure 3.4).

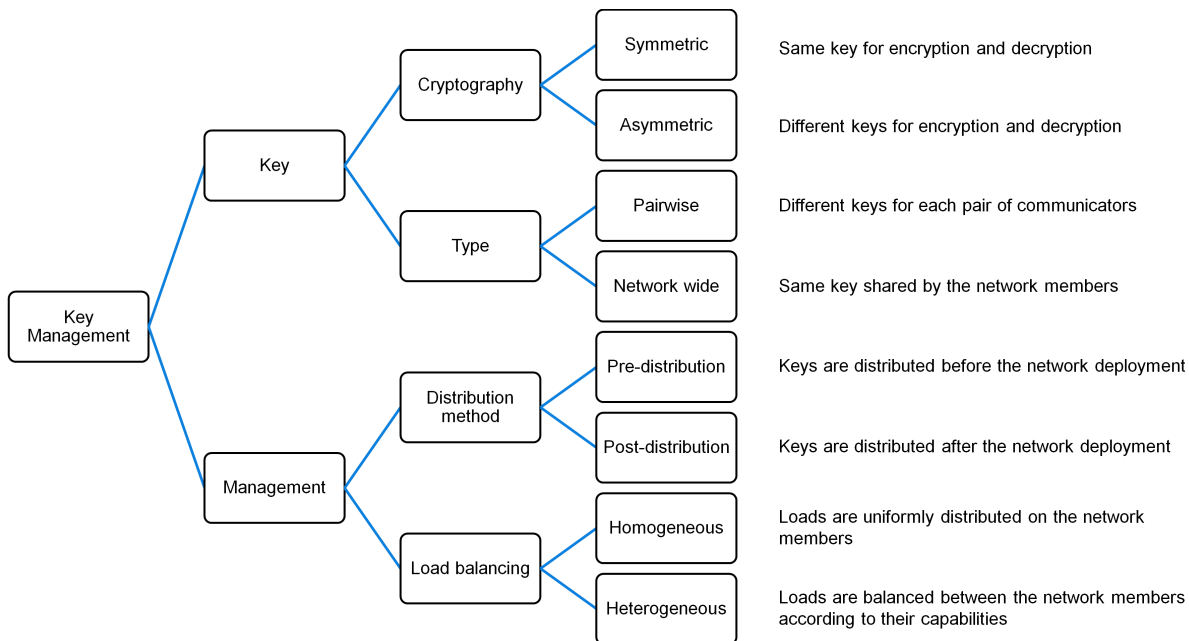


Figure 3.4 – Classification criteria.

#### 3.4.1.1 Key cryptography

Depending on whether the same key is used for encryption and decryption or not, the Key Management protocols are classified into two categories: symmetric and asymmetric schemes (Figure 3.5).

**Symmetric schemes:** These schemes involve the use of the same key for encryption and decryption (Figure 3.5a). They usually require affordable computing capacity and reasonable computing time. However, their effectiveness depends on the secrecy of the symmetric keys. The exchange of these keys generally require an amount of storage and communication growing with the network size. To sum up, Key Management protocols based on symmetric cryptography are efficient, but not scalable.

**Asymmetric schemes:** These schemes use two different keys for encryption and decryption (Figure 3.5b): a public key which may be disseminated widely and a private key which is known only to the owner. One is always calculated from the other so that if the first is used for encryption, the second can be used for decryption. Using asymmetric cryptography, no secret key exchange is required and a device only needs to store its own keys. However, the effectiveness of this method depends on the difficulty of guessing the private key from the public one. It is therefor based on computing power, which makes it resource intensive. To sum up, Key Management protocols based on asymmetric cryptography are scalable, but not efficient.

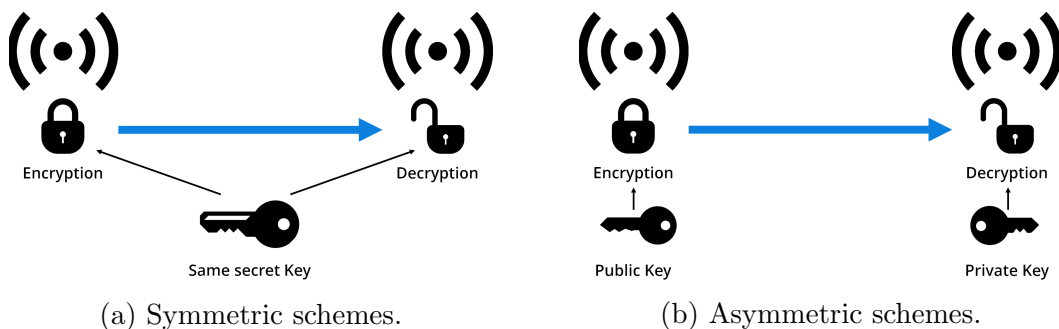


Figure 3.5 – Key cryptography.

### 3.4.1.2 Key type

Depending on whether distinct keys are used for each pair of communicators or the same key is shared by several members, the Key Management protocols are classified into two categories: pairwise and network wide key schemes (Figure 3.6).

**Pairwise schemes:** These schemes consist of using distinct keys for each pair of communicators (Figure 3.6a). They are resilient as capturing a member does not jeopardize the communication of the others. However, they are not well suitable for group communication as the same message must be encrypted and sent several times.

**Network wide schemes:** These schemes consist of using shared keys that are common to several network members (Figure 3.6b). They are efficient and scalable as they require little storage. However, they are not suitable for device-to-device communication as they lack resilience. Indeed, each device can decrypt the communication of the others.

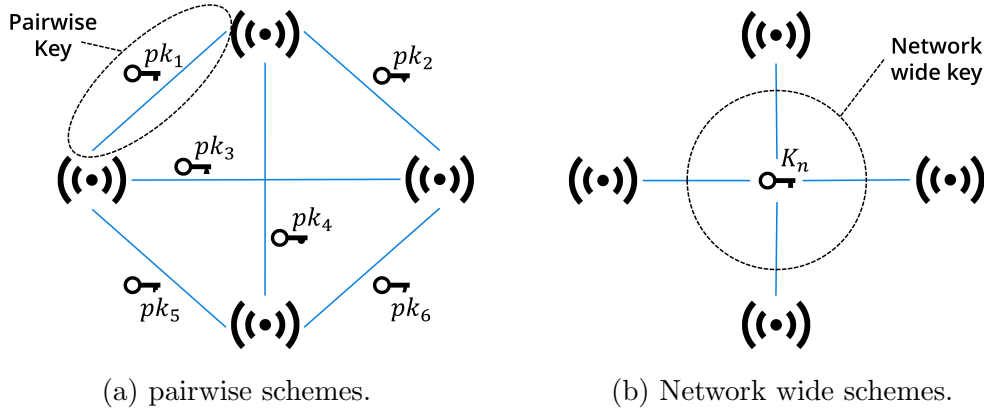


Figure 3.6 – Key Type.

### 3.4.1.3 Distribution method

Depending on whether the keys are distributed on the network members before deployment or after, the Key Management solutions are classified into two categories: pre-distribution and post-distribution schemes (Figure 3.7).

**Pre-distribution schemes:** These schemes involve the storage of keys in the devices' memory before their deployment (Figure 3.7a). The keys usually remain unchanged for the whole lifetime of the network. It is therefore difficult to add new devices to the network afterwards or revoke those that get compromised. Key Management protocols based on pre-distribution lack flexibility. They are not suitable for dynamic networks, whose members change frequently.

**Post-distribution schemes:** These schemes dynamically provide keys to the network members and update them when necessary (Figure 3.7b). To achieve this, a trusted authority is usually used. This centralized entity becomes nevertheless a single point of failure and the main target of attacks. If it fails, the entire system will stop operating and if it is attacked, the whole network will be compromised. Key Management protocols based on post-distribution pose a risk of unavailability and have a low level of resilience.

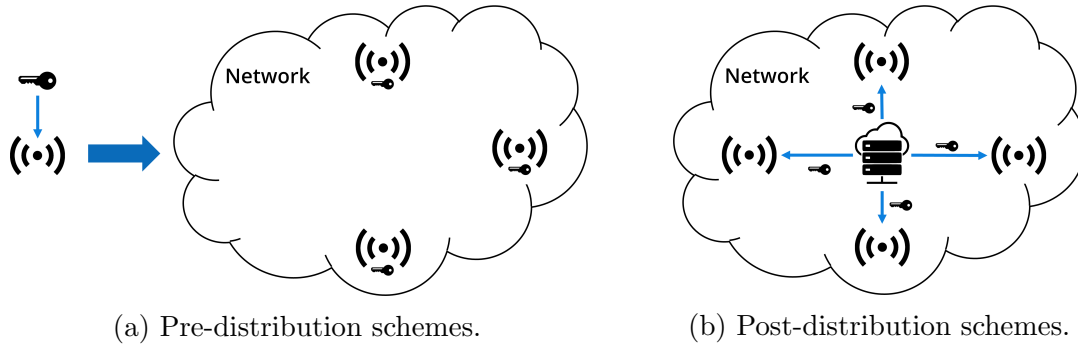


Figure 3.7 – Distribution method.

#### 3.4.1.4 Load balancing

Depending on whether the loads are uniformly distributed on the network members or balanced between them according to their capabilities, the Key Management protocols are classified into two categories: homogeneous and heterogeneous schemes (Figure 3.8).

**Homogeneous schemes:** These schemes uniformly distribute the loads on the network members and impose the same costs on all of them (Figure 3.8a). Thus, while a negligible amount of resource is sufficient for some, others will not have enough. Homogeneous Key Management protocols lack efficiency and scalability in heterogeneous networks.

**Heterogeneous schemes:** These schemes balance the overheads between the network members according to their capabilities (Figure 3.8b). By using a bit more of the resources of powerful devices, the constrained devices are more likely to support the overheads. Network performance is therefore improved and its lifetime increased. Heterogeneous Key Management protocols are efficient and highly scalable in heterogeneous networks.

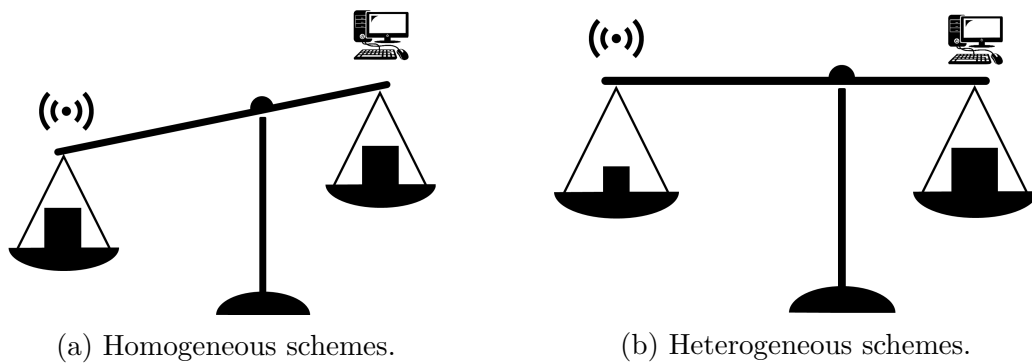


Figure 3.8 – Load balancing.

### 3.4.2 Proposed classification

To identify the challenges that related works still face in the IoT, we propose a classification of these protocols. Thus, we use the following notation to refer to a class of Key Management solutions:  ${}_{cry}^{typ}\mathcal{KM}_{dis}^{loa}$ . The variable “*cry*” refers to the key cryptography. It can take two values “*sym*” for symmetric protocols and “*asy*” for the asymmetric ones. The abbreviation “*typ*” indicates whether pairwise keys, “*pai*”, or a network wide key, “*net*”, is used. The variable “*dis*” can be replaced by “*pre*” for the protocols that are based on pre-distribution and by “*pos*” for post-distribution schemes. The abbreviation “*loa*” specifies whether the load balancing adopted by the scheme is homogeneous, “*hom*”, or heterogeneous, “*het*”. The notation “*hyb*” can be used, to replace any of the above-mentioned notations, when a scheme is based on a hybridization of two categories. Finally, when we refer to all the values of a given criteria we use the notation “*\**” (Table 3.2).

For example, the class  ${}_{hyb}^*\mathcal{KM}_{pos}^{hom}$  includes all the homogeneous and pre-distribution Key Management protocols that are based on a hybridization between symmetric and asymmetric cryptography. These protocols can use either pairwise keys or a network wide key, but not both at the same time.

Classification criteria	Possible choices	Notation
Key cryptography	Symmetric	${}_{sym}\mathcal{KM}$
	Asymmetric	${}_{asy}\mathcal{KM}$
	Symmetric and asymmetric	${}_{hyb}\mathcal{KM}$
	Symmetric or asymmetric	${}^*\mathcal{KM}$
Key type	Pairwise	${}_{pai}\mathcal{KM}$
	Network wide	${}_{net}\mathcal{KM}$
	Pairwise and network wide	${}_{hyb}\mathcal{KM}$
	Pairwise or network wide	${}^*\mathcal{KM}$
Distribution method	Pre-distribution	$\mathcal{KM}_{pre}$
	Post-distribution	$\mathcal{KM}_{pos}$
	Pre-distribution and post-distribution	$\mathcal{KM}_{hyb}$
	Pre-distribution or post-distribution	$\mathcal{KM}_*$
Load balancing	Homogeneous	$\mathcal{KM}^{hom}$
	Heterogeneous	$\mathcal{KM}^{het}$
	Homogeneous and heterogeneous	$\mathcal{KM}^{hyb}$
	Homogeneous or heterogeneous	$\mathcal{KM}^*$

Table 3.2 – Classification notations



In Table 3.3, we introduce a classification of the existing solutions based on the criteria and the notations presented above. Note that although 64 classes can be obtained using our classification, they have not all been used in the literature. We then discuss each of the existing classes and present its weaknesses.

Category	References	Discussion
$\mathcal{KM}_{sym}^{pai, hom, pre}$	[1, 2, 5, 19, 21, 26, 27, 32, 39, 40, 60, 81, 110, 130, 142–145]	[+] These approaches are efficient for secure device-to-device communication in static networks, whose members do not change frequently. [–] These approaches neither consider the group and multi-group communication nor the dynamic and heterogeneous nature of the IoT.
$\mathcal{KM}_{sym}^{pai, hom, post}$	[3, 8, 49]	[+] These approaches are efficient for secure device-to-device communication in dynamic networks.
$\mathcal{KM}_{sym}^{pai, hom, hyb}$	[30, 31, 48]	[–] These approaches do not consider the group and multi-group communication and lack scalability in heterogeneous networks containing limited-resource devices.
$\mathcal{KM}_{sym}^{pai, het, pre}$	[83]	[+] These approaches are efficient and scalable for secure device-to-device communication in heterogeneous networks. [–] These approaches do not consider neither the group and multi-group communication nor the dynamic nature of the IoT.
$\mathcal{KM}_{sym}^{pai, het, hyb}$	[41]	[+] These approaches are efficient and scalable for secure device-to-device communication in heterogeneous and dynamic networks. [–] These approaches do not consider the group and multi-group communication.
$\mathcal{KM}_{sym}^{net, hom, post}$	[45–47, 80, 120, 129, 133, 135, 138, 140]	[+] These approaches are efficient for secure group and multi-group communication in dynamic networks.
$\mathcal{KM}_{sym}^{net, hom, hyb}$	[148]	[–] These approaches do not consider the device-to-device communication and lack scalability in heterogeneous networks containing limited-resource devices.

$_{sym}^{net}\mathcal{KM}_{post}^{het}$	[79, 124]	<p>[+] These approaches are efficient and scalable for secure group and multi-group communication in heterogeneous and dynamic networks.</p> <p>[−] These approaches do not consider the device-to-device communication.</p>
$_{sym}^{hyb}\mathcal{KM}_{pre}^{hom}$	[58]	<p>[+] These approaches are efficient for secure device-to-device, group and multi-group communication in static networks.</p> <p>[−] These approaches do not consider the heterogeneous and dynamic nature of the IoT.</p>
$_{sym}^{hyb}\mathcal{KM}_{hyb}^{hom}$	[61, 89, 104, 145, 150]	<p>[+] These approaches are efficient for secure device-to-device, group and multi-group communication in dynamic networks.</p> <p>[−] These approaches lack scalability in heterogeneous networks containing limited-resource devices.</p>
$_{asy}^{pai}\mathcal{KM}_{post}^{hom}$	[29, 82, 103, 118, 121, 136, 137]	<p>[+] These approaches are scalable for secure device-to-device communication in dynamic networks.</p> <p>[−] These approaches do not consider neither group and multi-group communication nor the heterogeneous nature of the IoT. Also, being based on asymmetric encryption, they are not suitable for the IoT constrained devices.</p>
$_{asy}^{pai}\mathcal{KM}_{hyb}^{het}$	[105]	<p>[+] These approaches are scalable for secure device-to-device communication in dynamic and heterogeneous networks.</p> <p>[−] These approaches do not consider group and multi-group communication. Also, being based on asymmetric encryption, they are not suitable for the IoT constrained devices.</p>
$_{hyb}^{pai}\mathcal{KM}_{pre}^{hom}$	[111, 127]	<p>[+] These approaches are efficient for secure device-to-device communication in static networks.</p> <p>[−] These approaches neither consider the group and multi-group communication nor the dynamic and heterogeneous nature of the IoT.</p>
$_{hyb}^{pai}\mathcal{KM}_{post}^{het}$	[87, 92]	<p>[+] These approaches are efficient for secure device-to-device communication in heterogeneous and dynamic networks.</p> <p>[−] These approaches do not consider the group and multi-group communication.</p>

Table 3.3 – Classification of existing solutions

---

## 3.5 Key Management challenges in the IoT

Our literature review and the classification of the exiting solutions allow us to identify the challenges that are facing the Key Management in the IoT. We summarize these challenges in the following points:

- Most of the Key Management protocols consider either device-to-device or group and multi-group communication. As the IoT involves the three modes of communication, none of these schemes is suitable for it. Indeed, if the same key is used for all device-to-device communication, every network member will be able to decipher them. If several keys are used in group communication, the same message will be encrypted and sent several times. This will require additional calculation and communication and thereby more energy consumption. Finally, most of the existing schemes suffer from considerable overheads and are not suitable for the IoT constrained devices. Implementing different protocols will then be too heavy for them to handle.
- Most of the Key Management protocols proposed to secure device-to-device communication are based on pre-distribution. They are motivated by the fact that they do not require a third party to attribute secret keys to the network members. However, pre-distributed schemes are rigid as it is difficult to add new devices to the network after the deployment. They are therefore more suitable for static networks, which is not the case of the IoT.
- Most of the Key Management protocols based on post-distribution relay on a single entity to manage the keys. This entity becomes a single point of failure and the main target of attacks. If the central entity fails, the entire system will stop operating. Moreover, if this entity is attacked, the whole network will be compromised.
- Most of the Key Management protocols use the same parameters to secure all communications. As the IoT provides various services, communication within a service will be accessible to all devices even if they did not subscribe to it. Moreover, the capture of a member will jeopardize all services.
- Most of the Key Management protocols do not consider the heterogeneous nature of the IoT. They do not balance the loads between devices and impose the same costs on a powerful computer or a weak sensor. While a negligible part of the former's resources is used, those of the latter may not even be enough. Moreover, the few works, which are intended for heterogeneous networks, divide the devices into two classes only (powerful and constrained) and do not adapt to the network state.

### 3.6 Our contributions

In this thesis, we propose a novel Key Management protocol to secure the communication between the heterogeneous and dynamic IoT devices. We then show that our solution, of class  ${}_{hyb}^{hyb}\mathcal{KM}_{post}^{het}$ , is more suitable for the IoT than the existing solutions. Although this class best matches the IoT, we have not encountered it in the literature. An hybridisation of symmetric and asymmetric cryptography allows to take advantages of each and overcome its disadvantages. Combining pairwise and network wide keys also offers the possibility of secure the three modes of IoT communication. A post-distribution based solution consider the dynamic nature of the IoT. Finally, the IoT devices having different capabilities, the Key Management system must be heterogeneous. To design our solution, we proceed in stages and propose several major contributions:

- In chapter 4, we introduce a symmetric Key Management protocol for secure device-to-device communication in dynamic networks. Compared to the existing schemes of the class  ${}_{sym}^{pair}\mathcal{KM}_{post}^{hom}$ , our solution provides a good compromise between the IoT requirements: resilience, connectivity, mobility, flexibility, efficiency and scalability. To achieve this balance, the network members are uniformly distributed into logical sets. A device shares then a distinct pairwise key with each member of its set and a unique pairwise set key with the members of each of the other sets.
- In chapter 5, we enhance our protocol so that it also secures group communication in heterogeneous networks. Our new solution, of class  ${}_{sym}^{hyb}\mathcal{KM}_{post}^{het}$ , ensures the forward and backward secrecy and resists to collusion attacks. Furthermore, by balancing the loads between the heterogeneous devices according to their capabilities, our solution becomes more efficient and highly scalable.
- In chapter 6, we complete our solution so that it becomes of class  ${}_{hyb}^{hyb}\mathcal{KM}_{post}^{het}$ . In this final version, we first deal with the multi-group communication. We make sure that our solution guarantees the secure coexistence of several services in a single network. We also propose a decentralization of our protocol based on the blockchain technology and smart contracts. Finally, by combining symmetric and asymmetric cryptography, our solution securely distributes the Key Management on multiple entities. Thus, the system continues to operate when an entity fails and the compromise of a participant does not jeopardize the whole network.
- In chapter 7, we propose an implementation of our solution, considering the Contiki operating system and resource constrained IoT platforms, to experimentally complete the theoretical analyses.

### 3.6.1 Notations

Our solution can be hosted on servers (the cloud) or gateways to manage the keys, which are used by devices to secure communication. It can be implemented in a centralized or decentralized way. Thus, regardless of where and how it is implemented, we use the term Key Manager to refer to the implementation of our protocol on servers/gateways and the term node to refer to its implementation on devices (Figure 3.9).

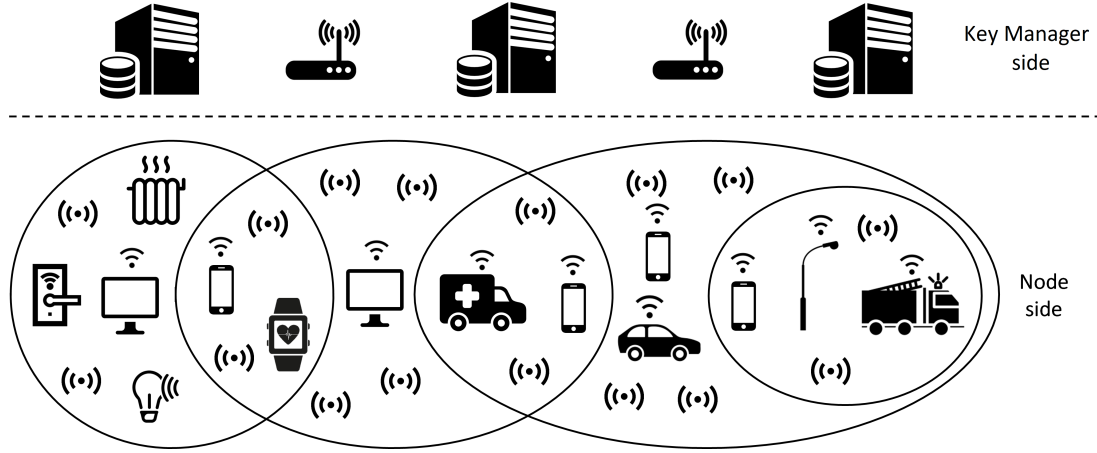


Figure 3.9 – IoT network (Key Manager and nodes).

The main notations, that are used in multiple sections of the manuscript, are summarized in Table 3.4.

Notation	Definition
$u, v, w$	Examples of nodes
$c_u$	The capability of the node $u$ in number of keys
$n$	The number of nodes in the network
$S, T, U$	Examples of sets (or subgroups)
$m_s$	The size of the set $S$
$mc_s$	The minimum capability of the set $S$
$p$	The number of sets in the network
$G, I$	Examples of groups
$N$	The network
$BP$	A Blockchain Participant
$r$	The number of $BPs$ in the network
$cp$	The consensus period
$ct$	Maximum temporary transactions
$KDF$	A Key Derivation Function

Table 3.4 – Summary of notations.

### 3.6.2 Application and threat model

Our solution can be used to secure communication in any Internet of Things application. It is indeed well suitable for dynamic and heterogeneous networks containing limited-resources devices. The motes on which we preformed the experiments are used in smart grids, smart homes, smart buildings, intelligent lighting systems and other IoT applications. A concrete example of an application for which the use of our solution would be interesting is smart cities. They indeed contain a huge number of heterogeneous devices (servers, computers, smartphones, gateways, sensors... etc) spread across the city. These devices can use the three communication modes of the Internet of Things (device-to-device, group and multi-group) to provide various services for the benefit of society (healthcare, intelligent transportation system... etc.).

A malicious device can be inside or outside the network [11] and may jeopardize the security of the three modes of communication (Figure 3.10). An outsider node can store the messages exchanged between the network nodes (group and multi-group communication) and decipher them when it joins the network. An evicted member can also pose a threat to the network, if it is still able to decipher the future communications. If a node or a *BP* inside the network is captured, it may try to decrypt the device-to-device communication of the other nodes. We assume that the blockchain is tamper proof (protected against P2P attacks such as eclipse or hijacking attacks). An attacker can not alter its content unless it has a capability that exceeds 51% of the overall network capacity.

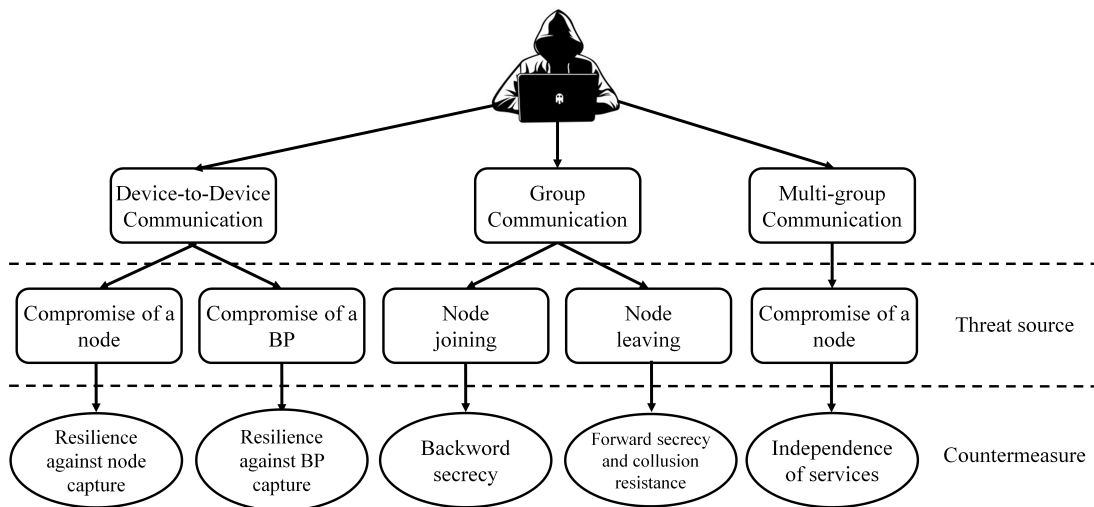


Figure 3.10 – Threat model and countermeasures

---

## 3.7 Conclusion

The purpose of this chapter was to present the Key Management in the IoT. We began by introducing the concept of Key Management and its important role in network security. This role includes the generation, storage, distribution and revocation of cryptographic keys. Next, we defined different criteria (the key cryptography, the key type, the distribution method and the load balancing) that can be used to classify the related works. We also presented some of the existing Key Management protocols and classified them according to the previously mentioned criteria. We discussed each of this categories by citing its advantages and disadvantages. The aim was to identify the challenges that the related works encounter. Finally, we summarized our contributions to overcome these challenges.

In the next chapter, we will present our first contribution. It is a novel symmetric Key Management protocol for secure device-to-device communication in dynamic networks. This first step of our solution belongs to the class  ${}_{sym}^{pair}\mathcal{KM}_{post}^{hom}$ .





# *Dynamic Key Management for Secure Device-to-Device Communication*

---

In this chapter, we propose a novel Key Management protocol for device-to-device communication in the IoT. Unlike most of the exiting schemes based on symmetric pairwise keys, our solution consider the dynamic nature of the IoT as it is based on post-distribution. Furthermore, compared to related works, our protocol provides a good compromise between the IoT requirements: resilience, connectivity, mobility, flexibility, efficiency and scalability. Indeed, we prove that our solution is resilient as the capture of a member compromises a negligible part of a large network. Moreover, we show that our scheme has a good network connectivity and allows node mobility. It is then efficient as it does not require additional calculation or communication costs on the network members. We also demonstrate that our protocol is scalable as storage cost on the network members does not significantly increase when the network gets larger. We finally show that our solution is flexible as it supports the dynamic deployment of nodes.

## 4.1 Introduction

In device-to-device communication (also called node-to-node or pairwise communication), an IoT device sends/receives unicast messages to/from another specific device [101]. An example of device-to-device communication is the vehicle-to-vehicle communication. It is a technology that allows two specific vehicles to exchange information about their speed and position. Thus, they can avoid crashes, ease traffic congestion and improve the environment [28]. The device-to-device communication in the IoT requires that the Key Management provides good resilience, connectivity, mobility, flexibility, efficiency and scalability. Although different protocols have been proposed to secure device-to-device communication, each of them presents its own limitations and weaknesses.

The existing solutions proposed for traditional Internet and static wireless sensor networks are not suitable for the IoT. They rarely consider the dynamic nature of its devices or their limited resources. Based on asymmetric cryptography, the Key Management protocols used in traditional Internet usually imply intensive computing, which makes them impractical on the IoT constrained devices [141]. The solutions proposed for static wireless sensor networks are lighter since most of them use symmetric cryptography. However, they generally store the keys in the device memories before deployment. This key pre-distribution makes it difficult to add new nodes afterwards in dynamic networks.

To address these issues, we propose a novel Key Management protocol, for device-to-device communication in the IoT, belonging to the class  $^{pair}_{sym}\mathcal{KM}^{hom}_{post}$ . Unlike most of the exiting schemes based on symmetric pairwise keys, our solution consider the dynamic nature of the IoT as it is based on post-distribution. Compared to related works, our solution provides a good compromise between the IoT requirements: resilience, connectivity, mobility, flexibility, efficiency and scalability. To achieve this balance, the nodes are uniformly distributed into logical sets. A device shares then a distinct pairwise key with each member of its set and a unique pairwise set key with the members of each of the other sets. We prove that our solution is resilient as the capture of a member compromises a negligible part of a large network. We also show that our scheme has a good connectivity and mobility. It is then efficient as it does not require additional calculation or communication costs on nodes. We finally demonstrate that our protocol is scalable as nodes' storage cost does not significantly increase when the network gets larger.

The remainder of this chapter is organized as follows. Section 4.2 presents related works. Section 4.3 introduces our solution. Section 4.4 presents the security analysis. Section 4.5 introduces the performance evaluation. Section 4.6 concludes the chapter.

## 4.2 Related Works

In this chapter, we are interested in device-to-device communication and symmetric cryptography. we then focus on the class  $_{sym}^{pai} \mathcal{KM}_*^*$ . Solutions belonging to this class can be classified into: deterministic, pure probabilistic and deployment knowledge based schemes.

### 4.2.1 Deterministic schemes

Deterministic schemes establish direct secure links between all communicators. The basic concept [27] consists of using a distinct pairwise key for each pair of devices. Other approaches were then proposed. Polynomial based protocols [8] use bivariate polynomials ( $f(x, y) = \sum_{(i,j)}^t a_{ij} x^i y^j$ ) instead of pairwise keys. These polynomials are chosen so that  $f(x, y) = f(y, x)$  and in each node  $i$  is stored  $f(i, y)$ . A pair of nodes  $(i, j)$  can calculate the shared key  $f(i, j)$ . Matrix based schemes [21, 39, 130] store, in each node  $i$ , the  $i^{th}$  row vector of a symmetrical matrix and the  $i^{th}$  column vector. Two nodes can exchange their columns and multiply them by their own secret row to get the shared pairwise key.

Deterministic schemes provide a perfect resilience, guarantee a total connectivity coverage and support node mobility. This is because each pair of nodes share a pairwise key. However, this imply that every device needs to store as many keys as there are nodes in the network. Although the work presented in [32] managed to reduce storage by half, it is still proportional to the network size. Furthermore, the larger is the network, the more vulnerable the polynomial and matrix based approaches are to compromise. This is because captured nodes can collaborate to recover the polynomial or the Matrix used to generate the keys. Deterministic schemes are not scalable and are not suitable for the IoT. Also, most of them lack flexibility as they are based on pre-distribution (Figure 4.1).

### 4.2.2 Pure probabilistic schemes

Pure probabilistic schemes are based on randomness to store fewer keys on nodes, without guaranteeing that each pair of nodes shares a key. The basic concept was introduced in [49]. It consists of generating a large pool of keys and to randomly distribute some of them (a key ring) to each node. Two neighboring nodes can then communicate only if they share a common key. Otherwise, they relay on intermediate nodes to establish secure links. Other protocols [1, 2, 41] were proposed to enhance this method. Using the Q-composite [27] scheme, nodes can communicate only if they share  $Q$  keys. Also, polynomial pool based schemes [26, 83, 110, 145] use a pool of polynomials instead of keys.

Probabilistic schemes are more scalable than the deterministic ones, since the storage on nodes is independent of the network size. They are nevertheless less resilient and efficient. Indeed, keys may be shared by more than two nodes, making them capable of deciphering each other's device-to-device communication. Furthermore, intermediate nodes may be necessary to establish secure links between the communicators that do not share a common key. This requires additional calculation and communication and thereby more energy consumption [144]. They also suffer from poor flexibility as most of them are based on key pre-distribution. Finally, probabilistic schemes do not provide a good connectivity and do not support node mobility (Figure 4.1). Some works tried to enhance the connectivity using, for example, the unital design theory [19] or system of equations [144]. Despite this, as long as they are probabilistic, the connectivity is rarely total.

### 4.2.3 Deployment knowledge based schemes

These schemes are neither deterministic nor purely probabilistic. They are based on the physical location of nodes to maximize the connectivity. Thus, to increase the probability of sharing keys, nodes are distributed into regional zones. Key rings are then assigned to them so that neighboring nodes share a maximum of keys. Like the other approaches, the deployment knowledge based schemes can use pairwise keys [30, 31, 40, 60], polynomials [81] or matrices [142].

Deployment knowledge based schemes are approximately as resilient and scalable as the pure probabilistic ones. They even provide a better network connectivity. However, they are not flexible and do not support node mobility. This makes them more suitable for static networks (Figure 4.1).

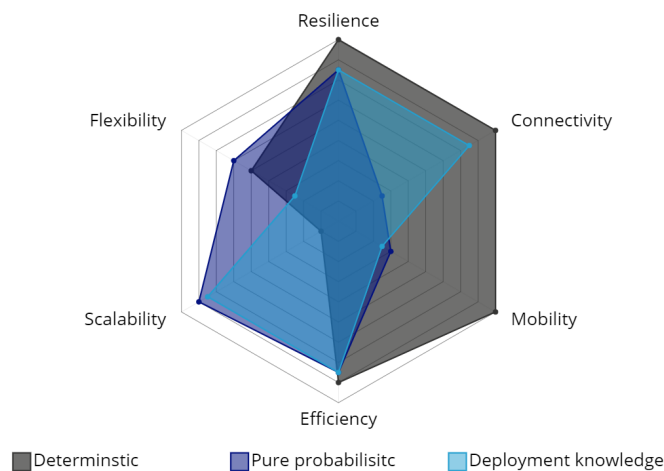


Figure 4.1 – Key Management approaches for secure device-to-device communication.

### 4.3 Our solution

Our literature review shows that none of the existing solutions, proposed to secure device-to-device communication, meets all the IoT requirements. We then propose a novel key Management protocol [68, 69] belonging to the class  $^{pair}_{sym}\mathcal{KM}_{post}^{hom}$ .

To improve the scalability of deterministic schemes without loss of efficiency, connectivity or mobility (unlike probabilistic schemes), our solution uniformly distributes the network members into logical sets. To each set  $S$  is associated a unique  $ID$ ,  $sid^S$ , and to each of its members  $u$  is assigned an  $ID$ ,  $nid_u$ , which is unique within  $S$ . A node shares then a distinct pairwise key with every member of its set and a unique pairwise set key with the members of each of the other sets. The scalability of the protocol is improved as nodes store fewer keys. It is important to note that these grouping is logical and independent of the application or the service that the devices are used for. Although nodes belonging to the same set are considered as cognates, they can be physically far from one another.

Although the members of a set share the same pairwise set key, we prove that our solution remains resilient against node capture for large network such as the IoT. Unlike deployment knowledge schemes, our protocol operates well regardless of the physical position of nodes. Moreover, as keys are dynamically distributed to the network members, when nodes join or leave the network, our solution is flexible (Figure 4.2).

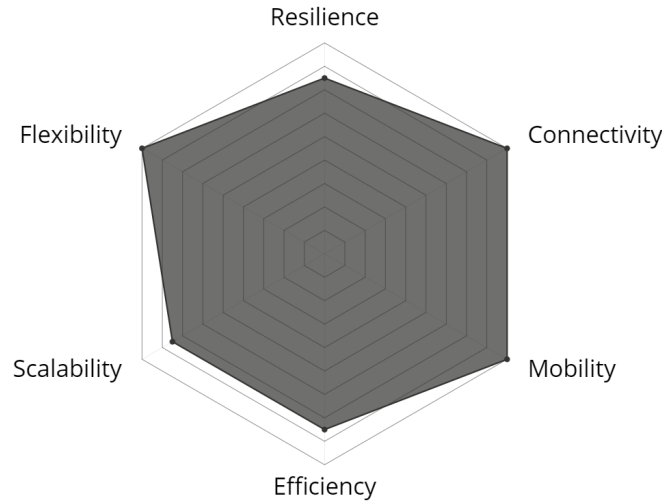


Figure 4.2 – Our device-to-device Key Management.

Since some keys are shared by several nodes, the Key Manager must ensure that they are known only by the current members. Thus, when a node joins or leaves the network, these keys are revoked and new ones are distributed to the remaining ones. This rekeying guarantees the backward and forward secrecy.

### 4.3.1 Classification of cryptographic keys

The keys managed by our solution can be classified into two types: Data Encryption Keys (*DEKs*) and Key Encryption Keys (*KEKs*). The *DEKs* are symmetric pairwise keys that are used by nodes to encrypt the data exchanged between them. The *KEKs* are used to secure the communications between the Key Manager and the nodes in order to protect the *DEKs* and thereby ensure the backward and forward secrecy. Let us consider a node  $u$  that belongs to a set  $S$ . The keys it holds are summarized in Table 4.1.

Key Type	Notation	Description
Data Encryption Key ( <i>DEK</i> )	$K_{u,v}^S$	This is a pairwise node key used by $u$ to secure the device-to-device communication with $v$ ( $v \in S$ ). A node has as many of these keys as there are members in its set
	$K^{S,T}$	This is a pairwise set key used by $u$ to secure the device-to-device communication with the members of the set $T$ ( $T \neq S$ ). A node has as many of these keys as there are sets in the network
Key Encryption Key ( <i>KEK</i> )	$K_u^S$	This is a node key used by $u$ to secure the communication with the Key Manager. It is known only by $u$
	$K^S$	This is a set key used to secure the communication with the Key Manager. It replaces the node key when the same message is sent to all the set members (for more efficiency). It is known only by the members of $S$
	$K_R$	This is a refresh key used for key update. It is not stored in the node memory

Table 4.1 – Classification of cryptographic keys.

Figure 4.3 shows an example of distribution of the nodes of a network  $N$  and the pairwise keys they share. Hereafter, the keys  $K_{u,v}^S$  and  $K_{v,u}^S$  are the same and can be used interchangeably. The same goes for the keys  $K^{S,T}$  and  $K^{T,S}$ .

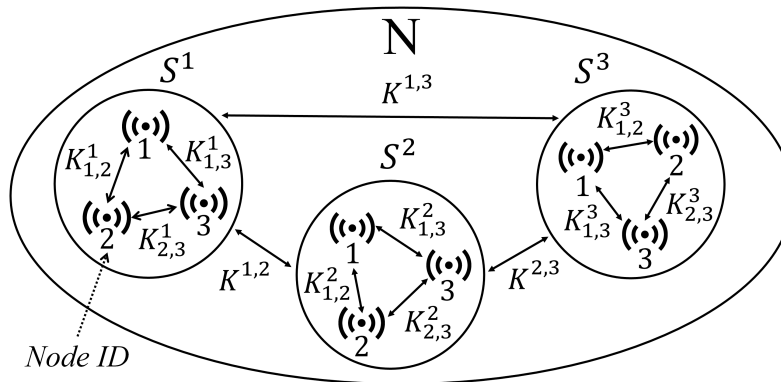


Figure 4.3 – Example of node distribution in a network  $N$ .

### 4.3.2 Hash functions

Using our protocol, nodes can share pairwise keys to reduce the storage cost. They are then able to decipher some messages that are not intended for them. To enhance the resilience of our solution without loss of scalability, we use hash functions (known to be efficiently computable). The aim is to create different keys from a single pairwise key.

#### 4.3.2.1 Zero-level approach

This term is used to refer to the basic approach that does not use hash functions. Although the other levels are more scalable, this approach is simpler. Therefore, unless otherwise stated, it is the Zero-level approach that is used in the rest of the manuscript.

#### 4.3.2.2 One-level approach

In this approach, a single hash function ( $H$ ) is used. Thus, instead of storing the pairwise set key  $K^{S,T}$ , a node  $u$  ( $u \in S$ ) stores its  $(nid_u - 1)^{th}$  hash,  $K_u^{S,T}$  (Formula 4.1). Since our solution is based on a symmetric approach, two nodes that wish to communicate must share the same key to securely communicate. Using hash functions, nodes belonging to two distinct sets may have different pairwise set keys. However, one of them can calculate the key known by the other. This is due to the fact that the keys they hold are calculated from the same key and using the same hash function.

$$K_u^{S,T} = H^{(nid_u-1)}(K^{S,T}) \quad (4.1)$$

To have a common communication key, the nodes rely on their  $IDs$ . Thus, knowing the  $IDs$  of each other, the node with the smallest one can use  $H$  to calculate the key of the other. Let us consider that the nodes  $u$  and  $v$  ( $v \in T$ ) wish to communicate. The first one stores  $K_u^{S,T} = H^{(nid_u-1)}(K^{S,T})$  and the second knows  $K_v^{S,T} = H^{(nid_v-1)}(K^{S,T})$ . If we assume that  $nid_u < nid_v$ ,  $u$  can calculate the  $(nid_v - nid_u)^{th}$  hash of its key and both nodes will have the same key (Formula 4.2).

$$\begin{aligned} K_v^{S,T} &= H^{(nid_v-1)}(K^{S,T}) \\ &= H^{((nid_v-1)-(nid_u-1))}(H^{(nid_u-1)}(K^{S,T})) \\ &= H^{(nid_v-nid_u)}(K_u^{S,T}) \end{aligned} \quad (4.2)$$

To fix ideas, let us consider the example of the Figure 4.3. We assume that the second node of the first set ( $u$ ) wishes to communicate with the third node of the second set ( $v$ ). The node  $u$  knows the key  $K_2^{1,2} = H(K^{1,2})$ , while  $v$  stores  $K_3^{1,2} = H^{(2)}(K^{1,2})$  (Figure 4.4). To have the same key and therefore be able to communicate,  $u$  must calculate the hash of the key it knows (Formula 4.3).

$$K_3^{1,2} = H^{(2)}(K^{1,2}) = H(H(K^{1,2})) = H(K_2^{1,2}) \quad (4.3)$$

Since hash functions are irreversible, the One-level approach ensures that nodes cannot decipher the messages exchanged between the other nodes that have smaller  $IDs$ . Although the resilience is improved, nodes are still able to decrypt the communications of those having bigger  $IDs$ . This has led us to propose the Two-level approach.

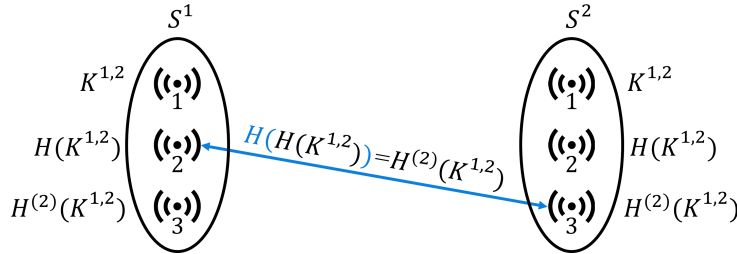


Figure 4.4 – One-level approach.

#### 4.3.2.3 Two-level approach

In this second approach, two different hash functions ( $H$  and  $H_2$ ) are used. A pairwise set key  $K^{S,T}$  is therefore split into two parts ( $\overleftarrow{K}^{S,T}$  and  $\overrightarrow{K}^{S,T}$ ), each hashed separately with one of the two functions. The node  $u$  ( $u \in S$ ) stores then the concatenation (noted by  $||$ ) of the  $(nid_u - 1)^{th}$  hash (calculated using  $H$ ) of the left part of the key and the  $(MAX - nid_u)^{th}$  hash (calculated using  $H_2$ ) of the right part (Formula 4.4). Note that  $MAX$  corresponds to the maximum number of nodes the sets may have.

$$K_u^{S,T} = H^{(nid_u-1)}(\overleftarrow{K}^{S,T}) || H_2^{(MAX-nid_u)}(\overrightarrow{K}^{S,T}) \quad (4.4)$$

Like the One-level approach, the hash functions and the  $IDs$  can be used by the nodes to calculate a common key. Knowing the  $IDs$  of each other, the node with the smallest one can apply  $H$  on the left part of its key and the other may apply  $H_2$  on the right part of its key. They will then have the same pairwise set key.



Let us consider that the nodes  $u$  and  $v$  ( $v \in T$ ) wish to communicate. The first one stores  $K_u^{S,T} = H^{(nid_u-1)}(\overleftarrow{K}^{S,T}) || H_2^{(MAX-nid_u)}(\overrightarrow{K}^{S,T})$  and the second knows  $K_v^{S,T} = H^{(nid_v-1)}(\overleftarrow{K}^{S,T}) || H_2^{(MAX-nid_v)}(\overrightarrow{K}^{S,T})$ . If we assume that  $nid_u < nid_v$ ,  $u$  can use  $H$  to calculate the  $(nid_v - nid_u)^{th}$  hash of the left part of the key it knows and  $v$  may use  $H_2$  to calculate the  $(nid_v - nid_u)^{th}$  hash of the right part (Formulas 4.5 and 4.6).

$$\overleftarrow{K}_v^{S,T} = H^{(nid_v-1)}(\overleftarrow{K}^{S,T}) \quad (4.5)$$

$$= H^{((nid_v-1)-(nid_u-1))}(H^{(nid_u-1)}(\overleftarrow{K}^{S,T}))$$

$$= H^{(nid_v-nid_u)}(\overleftarrow{K}_u^{S,T})$$

$$\overrightarrow{K}_u^{S,T} = H_2^{(MAX-nid_u)}(\overrightarrow{K}^{S,T}) \quad (4.6)$$

$$= H_2^{((MAX-nid_u)-(MAX-nid_v))}(H_2^{(MAX-nid_v)}(\overrightarrow{K}^{S,T}))$$

$$= H_2^{(nid_v-nid_u)}(\overrightarrow{K}_v^{S,T})$$

To fix ideas, let us consider the example of the Figure 4.3. We assume that  $MAX = 3$  and that the second node of the first set ( $u$ ) wishes to communicate with the third node of the second set ( $v$ ). The node  $u$  knows the key  $K_2^{1,2} = H(\overleftarrow{K}^{1,2}) || H_2(\overrightarrow{K}^{1,2})$ , while  $v$  stores  $K_3^{1,2} = H^{(2)}(\overleftarrow{K}^{1,2}) || \overrightarrow{K}^{1,2}$  (Figure 4.5). To have the same key and therefore be able to communicate,  $u$  can use  $H$  to calculate the hash of the left part of the key it knows and  $v$  may use  $H_2$  to calculate the hash of the right part of its key (Formulas 4.7 and 4.8).

$$\overleftarrow{K}_3^{1,2} = H^{(2)}(\overleftarrow{K}^{1,2}) = H(H(\overleftarrow{K}^{1,2})) = H(\overleftarrow{K}_2^{1,2}) \quad (4.7)$$

$$\overrightarrow{K}_2^{1,2} = H_2(\overrightarrow{K}^{1,2}) = H_2(\overrightarrow{K}_3^{1,2}) \quad (4.8)$$

This second approach is more resilient than the first one since it ensures that nodes cannot decipher, in addition to the messages exchanged between the members with smaller  $ID$ s, those exchanged by the nodes with larger  $ID$ s. This is because two different hash functions are used, one in ascending order of  $ID$ s and the other in descending order.

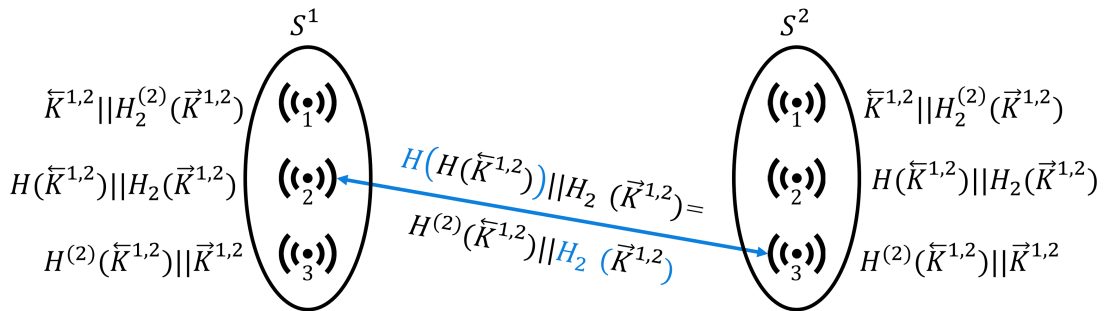


Figure 4.5 – Two-level approach.

### 4.3.3 Set management

The set management consists of distributing nodes on sets while minimizing the number of keys they store. The aim is to improve the protocol scalability without significant loss of resilience, efficiency, connectivity or mobility. In the following, we use the notations  $n$  and  $p$  to refer to the number of nodes and sets in the network, respectively. We also denote the number of members of a set  $S$  by  $m^S$ .

A member of  $S$  stores one secret key,  $m^S - 1$  pairwise node keys, one set key and  $p - 1$  pairwise set keys. Storage on nodes is therefore proportional to  $p + m^S$ . The problem consists then of creating sets and assigning nodes to them so as to satisfy:

$$\forall S, \min(p + m^S) \quad (4.9)$$

$$\sum_{S=1}^p m^S = n \quad (4.10)$$

To have the same number of keys stored on each network member, we opted for a uniform distribution (i.e.  $\forall S, m^S = m$ ). By replacing 4.10 in 4.9 and studying the monotony of the resulting function ( $f(p) = p + \frac{n}{p}$ ), we can easily show that storage is minimized when  $p = m = \sqrt{n}$ . The set management aims then to uniformly distribute the  $n$  nodes of the network into  $\sqrt{n}$  sets of  $\sqrt{n}$  members each (Figure 4.3).

#### 4.3.3.1 Assignment Algorithm

The Assignment Algorithm is run when nodes join the network and assigns them to the right sets. It takes as input  $n$ , the current number of network members, and outputs a set  $ID$  according to the input value. The algorithm manipulates then a list of sets,  $ls$ , of size  $p$ . Each of its items contains the  $ID$  of a set  $S$ ,  $sid^S$ , and its size,  $m^S$ .

When a node is authorized to join the network, the Assignment Algorithm starts by searching, in  $ls$ , a set  $S$  containing a number of nodes less than  $\sqrt{n}$ . If no set is found, a new one is created following the steps described in the next paragraph. Next, whether the set  $S$  is newly created or already exists, the algorithm assigns the joining node to it and updates  $ls$ . The steps of the Assignment Algorithm are described in Algorithm 1. Finally, the Key Manager renews the network security material, following the steps described in the node management section (See section 4.3.4).

**Set creation:** Creating a new set  $S$  consists of determining its  $ID$ ,  $sid^S$ , its key,  $K^S$ , and a pairwise set key for every set  $T$  of the network. Each of these pairwise set keys,  $K^{S,T}$ , is encrypted using the key of the set associated to it,  $K^T$ , and sent to its members (message  $CM$ ).

$$CM : KM \rightarrow T :< \{sid^S, K^{S,T}\}_{K^T} > (\forall T \in N, T \neq S)$$

---

**Algorithm 1:** Homogeneous Assignment Algorithm

---

**Input** :  $n$  = the number of network members

- 1 Search in  $ls$  a set  $S$  such that  $m^S < \sqrt{n}$ ;
  - 2 **if** no set is found **then**
  - 3     | Create a new set  $S$ ;
  - 4 **end**
  - 5 Assign the joining node to  $S$ ;
  - 6 Update  $ls$ ;
- 

#### 4.3.3.2 Reorder Algorithm

The Reorder Algorithm is run, after a node leaving, to reduce the number of sets by keeping the nodes distribution always uniform. It takes as input the network size,  $n$ , the percentage of merging,  $pcm$ , and tries to remove or merge sets when it is possible. To achieve this, the algorithm manipulates the same list  $ls$  used by the Assignment Algorithm.

When a node leaves a set  $S$ , the Algorithm starts by checking its new size. If  $S$  becomes empty, it is removed following the steps described below. On the other hand, if the size of  $S$  falls below a certain threshold, the algorithm searches in  $ls$  a set  $T$  to merge with  $S$ , following the steps described in the next chapter. The threshold corresponds to the product of  $pcm$  and  $\sqrt{n}$ . Note that the actual size of  $T$  must be less than the threshold as well. If it is the case, the two sets are merged. Also, the value of  $pcm$  must not exceed 50% so that the size of the resulting set does not exceed  $\sqrt{n}$ . Finally, the Key Manager renews the network security material, following the steps described in the node management section (See section 4.3.4). The steps of the Reorder Algorithm are described in Algorithm 2.

**Set removal:** Removing a set  $S$  consists of deleting its  $ID$ ,  $sid^S$ , its key,  $K^S$ , and all the pairwise set keys associated to it. The message  $RM$ , containing the  $ID$  of the set, is then sent to each remaining set  $T$  so that its members can remove the pairwise set key they share with the nodes of  $S$ .

$$RM : KM \rightarrow T :< \{sid^S\}_{K^T} > (\forall T \in N, T \neq S)$$

---

**Algorithm 2:** Homogeneous Reorder Algorithm

---

**Input** :  $n$  = the number of network members  
 $pcm$  = percentage of merging

```

1 if  $m^S = 0$  then Remove  $S$  ;
2 else
3   if  $m^S < pcm \cdot \sqrt{n}$  then
4     Find  $T$  such as  $m^T < pcm \cdot \sqrt{n}$ ;
5     if a set  $T$  is found then
6       Merge  $S$  and  $T$ ;
7     end
8   end
9 end
10 Update  $ls$ ;

```

---

### 4.3.4 Node management

In this section, we present the way in which our solution manages the keys upon a network change: node joining or leaving (Figure 4.6).

#### 4.3.4.1 Node joining

Let us consider a node  $u$  joining the network. The node is first assigned to a set  $S$ . The Key Manager generates then some new keys and updates some of the previously existing ones. The aim of this update is to ensure the backward secrecy. Indeed, if these keys are not updated and if the joining node  $u$  has stored the messages previously exchanged, it will be able to decipher some of them. Next, the Key Manager provides some nodes with the new keys and sends to others the elements allowing them to update some of the keys they hold. The process of node joining consists of the four following steps.

**Key generation:** The first step consists of determining the secret key,  $K_u^S$ , of the joining node  $u$ . After that, the Key Manager generates a pairwise node key,  $K_{u,v}^S$ , for each node  $v$  of the set  $S$ . It also determines the unique node ID,  $nid_u$ , associated to  $u$ .

**Key update:** The Key Manager starts by randomly generating  $K_R$ . Then, using it and a Key derivation function ( $KDF$ ), the Key Manager updates the set key of  $S$  and the pairwise set keys known by its members (Formulas 4.11 and 4.12, respectively).

$$K^{S+} = KDF(K^S || K_R) \quad (4.11)$$

$$K^{S,T+} = KDF(K^{S,T} || K_R), \forall T \in N \quad (4.12)$$

**Key distribution:** After the key generation and update are completed, the Key Manager distributes these new keys to the appropriate nodes. Thus, it sends to each node  $v$  of the set  $S$  the unicast message  $JM1$  encrypted by means of the node secret key,  $K_v^S$ . The message contains the  $ID$  of the joining node and the pairwise node key,  $K_{u,v}^S$ , associated to it. The Key Manager also broadcasts for each set  $T$  (including  $S$ ) the message  $JM2$  encrypted using  $K^T$ , the current set key of  $T$ . The message contains the  $ID$  of the set  $S$  and  $K_R$ . Finally, the Key Manager agrees with  $u$  on a temporary secret key (using a key agreement method). This key is then used to securely provide the joining node with its secret key, the new set key, the pairwise node keys to share with its cognates and all the new pairwise set keys associated to  $S$ . After the key distribution, the Key Manager discards  $K_R$ .

$$JM1 : KM \rightarrow v :< \{nid_u, K_{u,v}^S\}_{K_v^S} > (\forall v \in S)$$

$$JM2 : KM \rightarrow T :< \{sid^S, K_R\}_{K^T} > (\forall T \in N)$$

**Key installation:** When a member of  $S$ ,  $v$ , receives the messages  $JM1$  and  $JM2$ , it first decrypts them using its secret and set keys. Then, it installs  $K_{u,v}^S$  as the pairwise key to use for encrypting the communications with the joining node  $u$ . The node  $v$  also uses  $K_R$  and the  $KDF$  to update the set key and all the pairwise set keys it knows (Formulas 4.11 and 4.12, respectively). After that,  $v$  discards  $K_R$ . When a node  $w$ , not belonging to  $S$ , receives  $JM2$ , it first decrypts the message, using the current set key, and retrieves  $K_R$ . Then, using the  $KDF$ , it updates the pairwise set key it shares with the members of  $S$  (Formula 4.12). Once done,  $w$  discards  $K_R$ .

#### 4.3.4.2 Node leaving

A node  $u$  ( $u \in S$ ) can leave the network or be evicted when it get compromised. In both cases, the keys it knows must be revoked. The Key Manager removes then some of them and updates some others. The aim of this update is to ensure the forward secrecy. Indeed, if these keys are not updated, the leaving node will be able to decipher some of the future communications. Next, the Key Manager provides the network members with the elements allowing them to remove the keys that should be removed and to update those that must be updated. The process of node leaving consists of the four following steps.

**Key removal:** The Key Manager starts by removing the  $ID$  of the leaving node,  $nid_u$ , and its secret key,  $K_u^S$ . Next, it deletes all its pairwise keys,  $K_{u,v}^S$  ( $v \in S$ ,  $v \neq u$ ).

**Key update:** The Key Manager starts by randomly generating  $K_R$ . Then, using it and the  $KDF$ , the Key Manager updates the set key of  $S$ ,  $K^S$ , and all the pairwise set keys known by its members (Formulas 4.11 and 4.12, respectively).

**Key distribution:** After the key removal and update are completed, the Key Manager distributes the new keys to the appropriate nodes. Thus, it sends, to each node  $v$  of the set  $S$ , the unicast message  $LM1$  encrypted by means of the node key,  $K_v^S$ . The message contains the  $ID$  of the leaving node and  $K_R$ . The Key Manager also broadcasts, for each set  $T$  ( $T \neq S$ ), the message  $LM2$  to provide its members with  $K_R$ . The message  $LM2$  is encrypted using  $K^T$ , the current set key of  $T$ . The message  $LM2$  is not sent to the members of  $S$  because the leaving node  $u$  knows the set key  $K^S$ . The refresh key is therefore sent to the other members of  $S$  via the unicast message  $LM1$  instead. Finally, the Key Manager discards  $K_R$ .

$$LM1 : KM \rightarrow v : \{nid_u, K_R\}_{K_v^S} > (\forall v \in S, v \neq u)$$

$$LM2 : KM \rightarrow T : \{sid^S, K_R\}_{K^T} > (\forall T \in N, T \neq S)$$

**Key installation:** When a member of  $S$ ,  $v$ , receives  $LM1$ , it first decrypts the message, using its secret key  $K_v^S$ , and retrieves  $K_R$ . Then, it removes the pairwise key  $K_{u,v}^S$ , which was used for encrypting the communications with the leaving member  $u$ . The node  $v$  also uses the  $KDF$  to update the set key and all the pairwise set keys it knows (Formulas 4.11 and 4.12, respectively). Once done, the node  $v$  discards  $K_R$ . When a node  $w$ , not belonging to the set  $S$ , receives  $LM2$ , it first decrypts the message, using the current set key, and retrieves  $K_R$ . Then, using it and the  $KDF$ , the node updates the pairwise set key it shares with the members of the set  $S$  (Formula 4.12). Finally, the node  $w$  discards  $K_R$ .

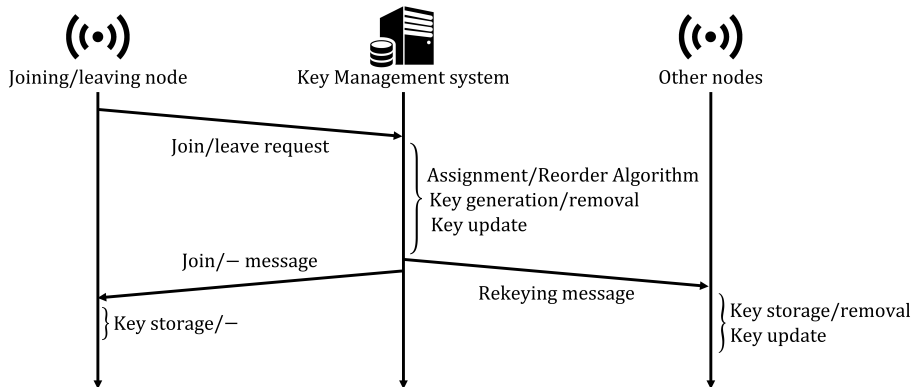


Figure 4.6 – Node joining and leaving.

## 4.4 Security analysis

In this section, we analyze the security of our solution. We then prove that it provides a good level of resilience. We now assume that the Key Manager itself is secure and that only the network nodes can be compromised. Furthermore, since some keys are shared by several nodes, we need to show that our solution fulfills the backward and forward secrecy. However, to avoid repeating the proofs, we will present them in the next chapter as it is devoted to group communication.

### 4.4.1 Theoretical analysis

According to [56], resilience is the measure of the impact of one captured node on the rest of the network. The issue is then to prove that, using our solution, this impact is negligible for large networks such as the IoT.

#### 4.4.1.1 Zero-level approach

Without hash functions, a node shares a distinct pairwise key with each of its cognates and a single pairwise key for each set of the network. It can decrypt, in addition to messages intended for it, those that are exchanged between its cognates and the other nodes. Figure 4.7 shows , among all possible communication links, those that a node can decrypt.

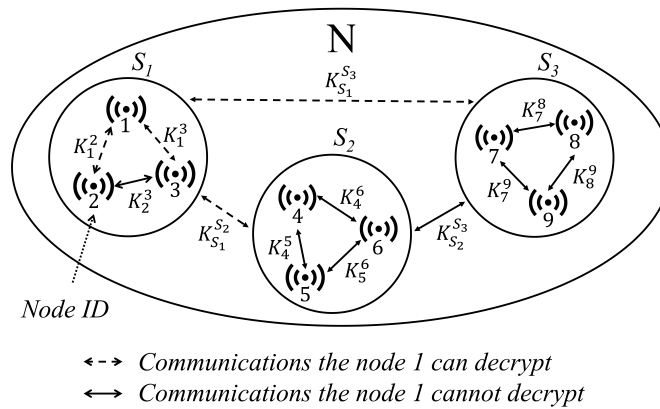


Figure 4.7 – An example of communications a node can decrypt.

**Lemma 4.1:** The number of links a node can decrypt is:

$$D_0 = (\sqrt{n} - 1)(n + 1) \quad (4.13)$$

**Proof:** A node can decrypt the communications linking it to the  $n - 1$  other network members as well as those between its  $\sqrt{n} - 1$  cognates and the  $n - \sqrt{n}$  other network members ( $D_0 = n - 1 + (\sqrt{n} - 1)(n - \sqrt{n})$ ).

**Lemma 4.2:** The number of links in a network of  $n$  nodes is:

$$T = \frac{n(n-1)}{2} \quad (4.14)$$

**Proof:** The total number of links is equal to the number of possible combinations that can be obtained by taking two nodes from  $n$  ( $T = C_n^2$ ).

**Proposition 4.1:** The percentage of compromised links due to a node capture is:

$$P_0 = \frac{2(n+1)}{n(\sqrt{n}+1)} \rightarrow 0, \text{ as } n \rightarrow \infty \quad (4.15)$$

**Proof:** From Lemmas 4.1 and 4.2 and the fact that  $P_0 = \frac{D_0}{T}$ , we deduce that  $P_0 = \frac{2(\sqrt{n}-1)(n+1)}{n(n-1)}$ .

**Proposition 4.2:** The capture of the whole network requires the compromise of all the network nodes.

**Proof:** Deciphering all the intra-set communications requires the knowledge of all the pairwise node keys associated to it. This is only possible if all the set members are captured. Also, deciphering all the inter-set communications requires the knowledge of all the pairwise set keys. This is only possible if at least a member of each set is captured.

#### 4.4.1.2 One-level approach

Using a hash function ensures that nodes cannot decipher the messages exchanged between the nodes that have smaller  $ID$ s (Figure 4.8a).

**Lemma 4.3:** The number of links a node  $u$  can decrypt is:

$$D_1 = (\sqrt{n} - 1)(n + 1 - (nid_u - 1)^2) \quad (4.16)$$

**Proof:** A node  $u$  cannot decrypt the links that connect each of its  $nid_u - 1$  elder cognates with the  $nid_u - 1$  members, that have smaller  $ID$ s, of each of the  $\sqrt{n} - 1$  other sets ( $D_1 = D_0 - (\sqrt{n} - 1)(nid_u - 1)^2$ ).



**Proposition 4.3:** The new percentage of links a compromised node  $u$  can decipher is:

$$P_1 = \frac{2(n+1 - (nid_u - 1)^2)}{n(\sqrt{n} + 1)} \rightarrow 0, \text{ as } n \rightarrow \infty \quad (4.17)$$

**Proof:** From Lemmas 4.2 and 4.3 and the fact that  $P_1 = \frac{D_1}{T}$ , we deduce that  $P_1 = \frac{2(\sqrt{n}-1)(n+1-(nid_u-1)^2)}{n(n-1)}$ .

#### 4.4.1.3 Two-level approach

The Two-level approach makes a member  $u$  unable to decrypt, in addition to the messages exchanged between the nodes with smaller  $ID$ s, those exchanged by the members with larger  $ID$ s (Figure 4.8b).

**Lemma 4.4:** The number of links a node  $u$  can decrypt is:

$$D_2 = (\sqrt{n} - 1)(n + 1 - (nid_u - 1)^2 - (\sqrt{n} - nid_u)^2) \quad (4.18)$$

**Proof:** A node  $u$  cannot decrypt the links that connect each of its  $\sqrt{n} - nid_u$  younger cognates with the  $\sqrt{n} - nid_u$  members, that have a larger  $ID$ , of each of the  $\sqrt{n} - 1$  other sets ( $D_2 = D_1 - (\sqrt{n} - 1)(\sqrt{n} - nid_u)^2$ ).

**Proposition 4.4:** The percentage of links a compromised node  $u$  can decipher is:

$$P_2 = \frac{2(n+1 - (nid_u - 1)^2 - (\sqrt{n} - nid_u)^2)}{n(\sqrt{n} + 1)} \rightarrow 0, \text{ as } n \rightarrow \infty \quad (4.19)$$

**Proof:** From Lemmas 4.2 and 4.4 and the fact that  $P_2 = \frac{D_2}{T}$ , we deduce that  $P_2 = \frac{2(\sqrt{n}-1)(n+1-(nid_u-1)^2-(\sqrt{n}-nid_u)^2)}{n(n-1)}$ .

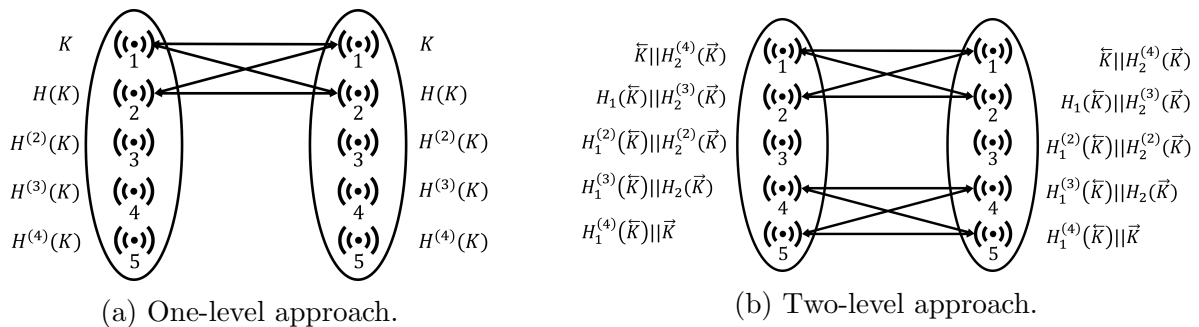


Figure 4.8 – An example of communications the node 3 cannot decrypt.

### 4.4.2 Comparison

We start by comparing the resilience of the three approaches we propose (Zero-level, One-level and Two-level). After that, we compare our solution to related works.

To compare the resilience of the three approaches, we consider a network of 10000 members, divided into 100 sets of 100 members each. We then compare the variation of the three percentage  $P_0$ ,  $P_1$  and  $P_2$  according to the node  $ID$ . The results (Figure 4.9a) show that the use of hash functions reduces the rate of compromised links. If a Zero-level approach is used, the percentage is maximum and constant regardless of the node  $ID$ . This method is still the best in terms of calculation cost. If a One-level approach is used, the bigger the  $ID$  of the captured node, the smaller the percentage of compromised links. This method is therefore interesting if we trust the old nodes more than the new ones (Assuming  $IDs$  are assigned in the order the nodes joined the network). Finally, the Two-level approach provides the best resilience regardless of the node  $ID$ .

Now, we compare our protocol to the deterministic scheme presented in [32]. Providing a perfect resilience (at the expense of scalability), none of the other solutions can provide a better level of resilience. This perfect resilience is achieved by using a distinct pairwise key for each pair of nodes. Thus, a captured node can only decipher the  $n - 1$  communications linking it to the other network members. The percentage of compromised links due to a node capture is then equal to  $\frac{2(n-1)}{n(n-1)} = \frac{2}{n}$  (Proportional to  $\frac{1}{n}$ ). On the other hand, we showed that, using our Zero-level approach, this percentage is equal to  $P_0$  (Proportional to  $\frac{1}{\sqrt{n}}$ ). Figure 4.9b shows that the value of  $P_0$  is negligible for large networks such as the IoT. It is even comparable to the rate provided by the perfectly resilient approaches. We also showed that the compromise of the whole network requires the capture of all its members. Our solution offers then a good level of resilience.

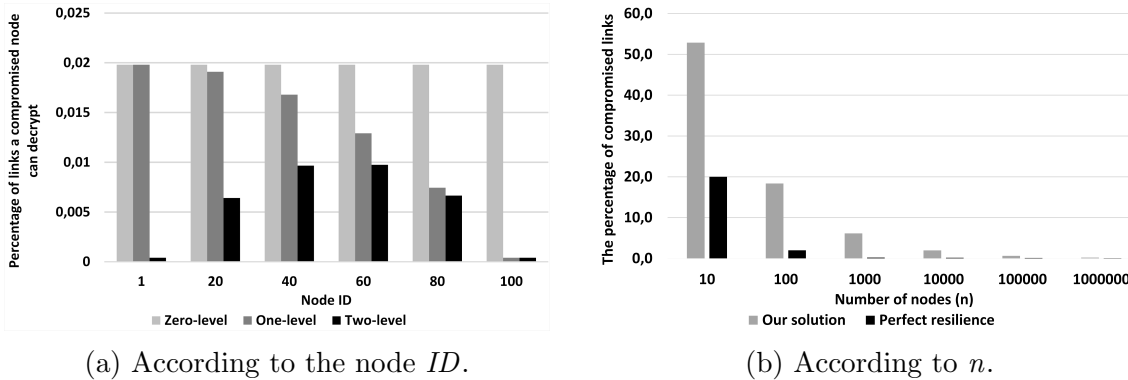


Figure 4.9 – Variation of the percentage of compromised links.

## 4.5 Performance evaluation

In this section, we evaluate the performance of our solution. After showing that it offers a level of resilience comparable to that provided by deterministic schemes, we prove that it is as scalable as probabilistic schemes (pure probabilistic and deployment-knowledge based approaches) without significant loss of efficiency, connectivity, mobility or flexibility.

### 4.5.1 Theoretical analysis

The Key Management can be hosted on the cloud (servers) or implemented on gateways, which have plentiful of resources. It is then more important to make the costs affordable on the devices as most of them suffer from a lack of resources. We begin by briefly analyzing the costs of our solution on the Key Manager before detailing them on devices.

#### 4.5.1.1 Overheads on the Key Manager

We start by analyzing the overheads of our solution on the Key Manager side.

**Property 4.1:** The communication overhead on the Key Manager is  $O(\sqrt{n})$ .

**Proof:** Regardless of the rekeying operation performed (such as a node joining  $S$  or a node leaving  $S$ ), the Key Manager sends a unicast message to each of the  $\sqrt{n}$  members of  $S$  and broadcasts a message for each of the other  $\sqrt{n} - 1$  sets, in the worst case. The Key Manager then sends a total number of messages proportional to  $\sqrt{n}$ .

**Property 4.2:** The calculation overhead on the Key Manager is  $O(\sqrt{n})$ .

**Proof:** Regardless of the rekeying operation performed (such as a node joining  $S$  or a node leaving  $S$ ), the Key Manager updates the keys which are or will be known by the node in question. The calculation overhead is therefore proportional to the storage cost on nodes, which will be proven to be of the order of  $\sqrt{n}$  in the next section. Also, the set management algorithms browse the list of sets in the worst case. They therefore have a complexity proportional to  $\sqrt{n}$ .

**Property 4.3:** The storage overheads on the Key Manager is  $O(n)$ .

**Proof:** The number of nodes is more important than that of sets. Thus, if we choose not to store the pairwise node keys (used to secure communication between the nodes) in the Key Manager's memory, the largest number of keys to save will be that of the node secret keys. The Key Manager will then store a number of keys proportional to  $n$ .

**Discussion:** The communication and calculation costs on the Key Manager are  $O(\sqrt{n})$ . The storage, on the other hand, is  $O(n)$ . Considering the significant improvement (presented in the next section) that our solution provides on the node side, the costs on the Key Manager are very reasonable. Indeed, as the Key Manager has usually plentiful of resources, we aimed to make the costs more affordable on the nodes.

#### 4.5.1.2 Overheads on the nodes

Now, we study the overheads of our solution on the device side.

**Property 4.4:** The communication overhead on the nodes is  $O(1)$ .

**Proof:** Regardless of the rekeying operation performed (such as a node joining  $S$  or a node leaving  $S$ ), a node receives a constant number of messages. The communication cost on nodes is therefore independent of the network size.

**Property 4.5:** The calculation overhead on the nodes is  $O(\sqrt{n})$ .

**Proof:** Regardless of the rekeying operation performed (such as a node joining  $S$  or a node leaving  $S$ ), a node updates all the keys it knows, in the worst case. The calculation cost on the nodes is therefore proportional to the storage, which will be proven to be of the order of  $\sqrt{n}$  in the next proof.

**Property 4.6:** The storage overhead on the nodes is  $O(\sqrt{n})$ .

**Proof:** Using our solution, a node knows a secret key,  $\sqrt{n} - 1$  pairwise node keys, a set key and  $\sqrt{n} - 1$  pairwise set keys. It then stores in total  $2 \cdot \sqrt{n}$  keys.

**Discussion:** The communication cost on the nodes is  $O(1)$ . The calculation and storage overheads, on the other hand, are  $O(\sqrt{n})$ . Since communication is the operation that consumes the most energy, our solution is efficient and highly scalable.

### 4.5.2 Comparison

After showing that our solution provides a good level of resilience, let us prove that it meets the other IoT requirements: scalability, connectivity, mobility, efficiency and flexibility. Thus, we compare our solution to the existing Key Management protocols proposed to secure device-to-device communication. We consider PKS [32] and Kronecker [130] as deterministic schemes and UKP [19] and Trade [110] as probabilistic approaches.

#### 4.5.2.1 Scalability

Although having a perfect resilience, the storage cost of the pairwise key schemes, in general, and PKS, in particular, is  $O(n)$ . Indeed, a node has to store a pairwise key for each of the other network members. On the other hand, Kronecker and Trade has a storage proportional to  $O(\sqrt{n})$ . For the other probabilistic schemes (e.g. UKP), it is difficult to deduce the storage from the network size as it depends on other parameters (pool size, network connectivity, deployment knowledge...etc). Despite this, the authors in [49] show that for a probabilistic scheme to establish almost certain connectivity for 10000 nodes, 250 keys out of a pool of 100000 keys have to be stored on the nodes. Our solution has a storage proportional to  $O(\sqrt{n})$ . Thus, for the same number of nodes and with a total connectivity coverage, it requires the storage of only 100 keys on the nodes. Figure 4.10 shows that our solution stores fewer keys than the pairwise key schemes and can operate on larger networks of compromised nodes such as the IoT. It even provides a level of scalability comparable to Kronecker and Trade.

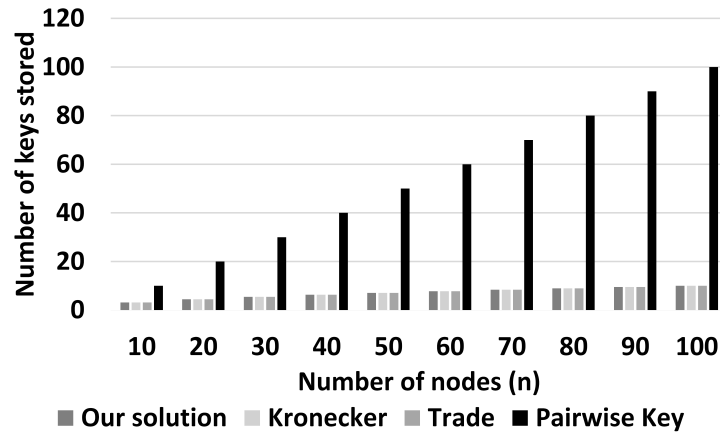


Figure 4.10 – Variation of nodes' storage overhead according to  $n$ .

#### 4.5.2.2 Connectivity

Although being scalable, the probabilistic schemes, mentioned above, suffer from poor connectivity. The probability that two neighboring nodes share a common key does not exceed 0.25 in Trade, while in UKP it is approximately lower bounded by 0.632. Using our solution, each pair of communicators share a pairwise node or a set key and can establish a direct secure link without relying on intermediate nodes. This is always possible even if the nodes move. Our solution provides then a good connectivity and mobility.

### 4.5.2.3 Mobility

Although deployment knowledge schemes [31, 81, 142] provide good connectivity, they are based on nodes' location. Our solution operates well regardless of the position of nodes. It then provides a better mobility, which makes it more suitable for mobile networks such as the IoT.

### 4.5.2.4 Efficiency

When connectivity and mobility are low, neighboring nodes may relay on intermediate nodes to establish secure links. The path length represents the number of intermediate nodes separating two communicators. The results presented in [49] give an overview about the average path length between two nodes using a probabilistic scheme. It is important to note that the longer the path, the more the communication between nodes requires additional calculation and communication. This reduces the efficiency of the protocol.

Unlike most of the probabilistic schemes (e.g. Trade and UKP), our solution has good connectivity and mobility. Figure 4.11 shows the large gap between the value of the path length using a probabilistic scheme and our solution, regardless of the network size. Furthermore, our solution stores fewer keys on nodes than the pairwise key schemes (e.g. PKS) (Figure 4.10). The communication and calculation costs are the same as Kroneker, while they are of the order of  $O((\log(n))^2)$  and  $O(\log(n))$ , respectively, in Trade. Note that communication is the operation that consumes the most energy. On top of that, our solution is based on symmetric cryptography. It is therefore efficient and well suitable for the IoT devices.

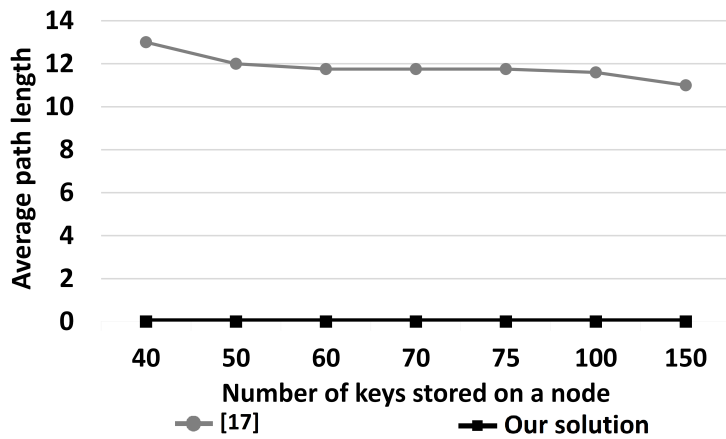


Figure 4.11 – Variation of the average path length according to the size of the key ring.

#### 4.5.2.5 Flexibility

Unlike all the above-mentioned schemes based on pre-distribution, our solution supports the dynamic deployment of nodes. Indeed, we previously showed that nodes can join and leave the network at any time without jeopardizing its security. Our protocol is then more flexible and suitable for dynamic networks such as the IoT.

## 4.6 Conclusion

The purpose of this chapter was to present a novel Key Management protocol for device-to-device communication in the IoT. Our solution provides a good compromise between the IoT requirements (resilience, connectivity, efficiency, scalability and flexibility) compared to the existing device-to-device protocols. This balance is achieved by uniformly distributing the network members into logical sets. A node shares then a distinct pairwise key with each member of its set and a unique pairwise set key with the members of each of the other sets. We proved that the capture of a member compromises a negligible part of a large network. Our solution is therefore resilient. Next, we showed that it has a good connectivity and allows node mobility. It is then efficient as it does not require additional calculation or communication. We also demonstrated that storage on nodes does not significantly increase when the network gets larger. Thus, our scheme is scalable. We finally showed that it is flexible.

In the next chapter, we will introduce our second contribution. It is a novel versatile Key Management protocol for secure device-to-device and group communication in heterogeneous and dynamic networks. This new solution belongs to the class  $\mathcal{KM}_{sym}^{hyb}{}_{post}^{het}$ .





# *Heterogeneous Key Management for Secure Group Communication*

---

In this chapter, we propose a novel versatile Key Management protocol for the IoT. Unlike most of the exiting schemes, our solution secures both device-to-device and group communication. It also considers the heterogeneous nature of the IoT. Thus, by using a bit more of the resources of powerful devices, our solution becomes much lighter for the constrained ones. This significantly improves the network performance and increase its lifetime. We then show that our solution ensures the forward and backward secrecy. Indeed, nodes can securely join and leave the network at any time. We also prove that our protocol resists to collusion attacks, as multiple evicted nodes can not cooperate to regain access to the network. Finally, we show that, by balancing the loads between the heterogeneous devices according to their capabilities, our solution is both efficient and highly scalable.

## 5.1 Introduction

In group communication, an IoT device sends/receives broadcast messages to/from the members of the group to which it belongs [129]. This device must have previously joined the group legally. It can also voluntarily leave it afterwards or be evicted if it gets compromised. An example of group communication is the Vehicle-to-Everything communication. It consists of allowing a vehicle to communicate with all the nearby devices (e.g. cars, bicycles, public lighting). The aim is to make the vehicle sense its environment and take the right decision [28]. The group communication requires that the Key Management ensures backward and forward secrecy, collusion resistance, efficiency and scalability. Although different protocols have been proposed to secure group communication, each of them presents its own limitations.

Most of the existing solutions do not consider the heterogeneous nature of the IoT devices or their limited resources. The Key Management protocols used in traditional Internet are based on asymmetric cryptography. Therefore, they are usually compute intensive and not well suited for the IoT constrained devices [141]. The solutions proposed for wireless sensor networks generally use symmetric cryptography and are lighter. However, they generally impose the same costs on the powerful devices and the weak ones. Thus, while a negligible part of the former's resources is used by the protocol, those of the latter may not even be enough. This exhausts the resources of the constrained devices, which can significantly degrade the network performance and shorten its lifetime. It may also happen that some constrained nodes cannot support the overheads at all, while others can handle much more.

To address these issues, we enhance our Key Management protocol so it secures, in addition to device-to-device communication, the group communication. This new solution belongs to the class  ${}_{sym}^{hyb}\mathcal{KM}_{post}^{het}$ . Unlike most of the existing schemes, our protocol considers the heterogeneous nature of the IoT. We show that, by balancing the loads between the heterogeneous devices according to their capabilities, our solution is both efficient and highly scalable. We also prove that our solution ensures the forward and backward secrecy. Indeed, nodes can securely join and leave the network at any time. Finally, we show that our protocol resists to collusion attacks, as multiple evicted nodes can not cooperate to regain access to the network.

The remainder of this chapter is organized as follows. Section 5.2 presents related works. Section 5.3 introduces our solution. Section 5.4 presents the security analysis. Section 5.5 introduces the performance evaluation. Section 5.6 concludes the chapter.

## 5.2 Related Works

The solution we present in this chapter secures both device-to-device and group communication. However, as the former was detailed in the previous chapter, we focus in this section on the latter. We then consider the class  $\mathcal{KM}_{*}^{net, sym}$ . Solutions belonging to this class are usually based on: tree structures, combinatorial optimization or batch rekeying.

### 5.2.1 Tree based schemes

The Logical Key Hierarchy (LKH) [135, 138] consists of using a tree structure to reduce the communication cost during the process of rekeying. The root of the tree corresponds to the group key, its leaves to the members' secret keys and the other nodes to intermediate keys (Figure 5.1). Each member stores the keys forming its branch. When a device joins or leaves the group, the server replaces only the keys it knows. The rekey message contains each of the new keys encrypted by its respective children. In the case of a binary tree, nodes' storage cost will be proportional to  $O(\log_2(n))$  and the size of the rekey message to  $O(2\log_2(n))$ . The One-way Function (OFT) protocol [120] was then proposed. It uses a one-way function to reduce the size of the rekey message to  $O(\log_2(n))$ . Both CASMA and GROUPIT protocols aim to deal with the dynamicity of IoT environments. While the former divides the network into multiple zones each implementing LKH [63], the latter combines LKH with the Chinese Remainder Theorem [75].

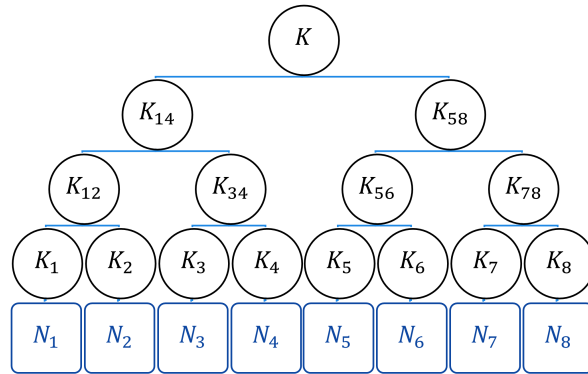


Figure 5.1 – Tree based schemes.

The Tree based schemes are usually secure as they guarantee the backward and forward secrecy and are resistant to collusion attacks. They are also reasonably efficient and provides a good scalability. Nonetheless, these schemes rarely consider the heterogeneous nature of the IoT (Figure 5.2).

### 5.2.2 Combinatorial optimization based schemes

The Exclusion Basic System (EBS) scheme is based on combinatorial optimization. It aims to make it possible to choose a compromise between the number of keys stored on nodes and that of messages exchanged during the rekeying process. The idea was first introduced in [45]. Other protocols were then proposed to improve the efficiency and the collusion resistance. The protocols GKIP [47] and SHELL [140], for example, are based on nodes deployment knowledge to achieve this, while LOCK [46] uses two layers of EBS.

The EBS based schemes ensure the backward and forward secrecy. They are efficient and scalable. Nevertheless, they are generally vulnerable to collusion attacks and do not consider the heterogeneous nature of the IoT (Figure 5.2).

### 5.2.3 Batch rekeying based schemes

Most of the exiting dynamic Key Management schemes are based on individual rekeying, i.e. they rekey the group after each join or leave request. For more efficiency, the batch rekeying based schemes [79, 80, 133] were proposed. The main idea is to gather several requests and to periodically rekey the group. The aim is to reduce the rekeying overheads.

Batch rekeying based schemes are more efficient than those based on individual rekeying. However, a new node has to wait until the end of the period to actually join the network. More importantly, as long as the group key has not been replaced yet, a leaving or an evicted member can still decipher the communications. Forward secrecy is then not totally guaranteed (Figure 5.2).

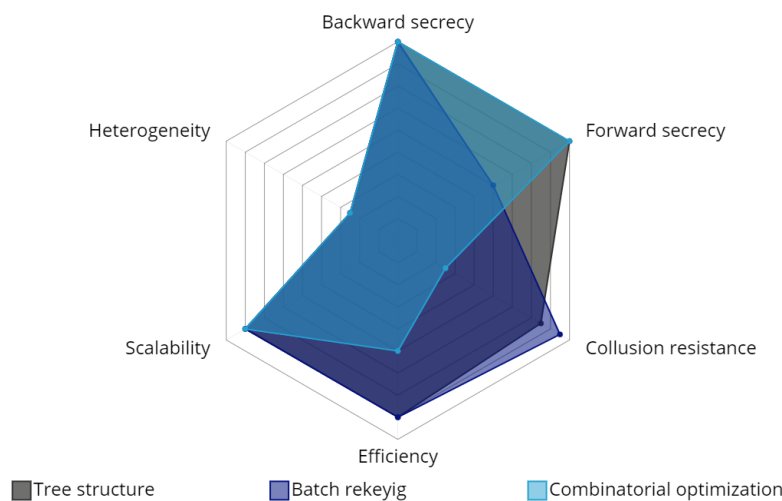


Figure 5.2 – Key Management approaches for secure group communication.

## 5.3 Our solution

Our literature review shows that most of the existing Key Management schemes do not consider the heterogeneous nature of the IoT. Moreover, they are usually intended either to device-to-device or group communication, and rarely to both of them. For these reasons we propose a novel versatile and heterogeneous Key Management protocol [69, 70] belonging to the class  $\mathcal{KM}_{sym}^{hyb} \mathcal{M}_{post}^{het}$ . It is an improvement of our solution (presented in the previous chapter) that considers the heterogeneous nature of the IoT. It also secures the group communication, in addition to the device-to-device communication.

Using our solution, the network (or the group  $G$ ) is partitioned into logical subgroups (or sets). This partitioning is logical and transparent to the application layer. Nodes belonging to the same subgroup can be physically far from each other. The objective behind this is rather to reduce the protocol overheads and to efficiently rekey the group when necessary. We previously showed that the costs of our solution on nodes depend on the size of their subgroups. Thus, we propose a novel heterogeneous subgrouping. According to their capabilities, nodes of a heterogeneous network are distributed into subgroups having different sizes to balance the loads between them (Figure 5.3). The aim is to reduce the costs on constrained nodes. The network performance is then improved and its lifetime increased. Moreover, the constrained nodes are more likely to support the overheads when the network gets larger.

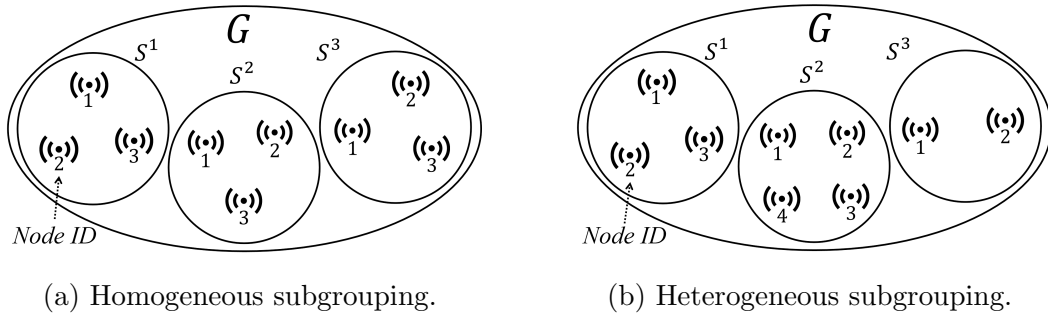


Figure 5.3 – Example of a group partitioning.

Furthermore, as the IoT devices may use the device-to-device and group communication, we enhance our solution so that it considers the two of them. We then show that our solution ensures the forward and backward secrecy. Indeed, nodes can securely join and leave the network at any time. We also prove that our protocol resists to collusion attacks, as multiple evicted nodes cannot cooperate to regain access to the network. Finally, we show that, by balancing the loads between the heterogeneous devices according to their capabilities, our solution is both efficient and highly scalable (Figure 5.4).

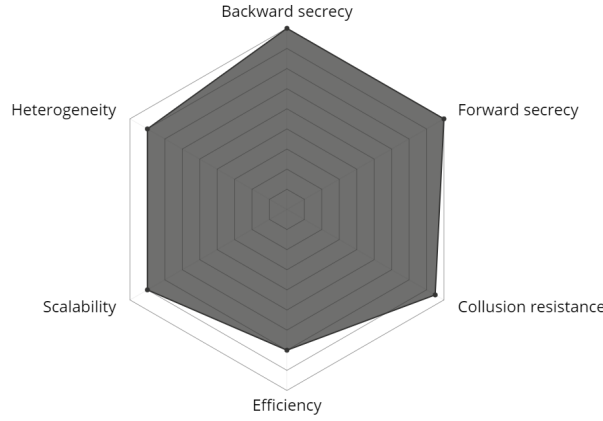


Figure 5.4 – Our group Key Management.

### 5.3.1 Classification of cryptographic keys

The keys managed by our solution can be classified into two types: Data Encryption Keys (*DEKs*) and Key Encryption Keys (*KEKs*). The *DEKs* are symmetric pairwise keys that are used by nodes to encrypt the data exchanged between them. They include the pairwise node keys (used to secure device-to-device communication between the nodes belonging to the same subgroup), the pairwise subgroup keys (used to secure the device-to-device communication between the nodes belonging to distinct subgroups) and the network wide key (or the group key,  $K_{(G)}$ , that is known by all the group members and used by them to secure the group communication). On the other hand, the *KEKs* are used to protect the *DEKs* and thereby ensure the backward and forward secrecy. They include the node keys (used by nodes to secure the communication with the Key Manager) and the subgroup keys (which replaces the node keys when the same message is sent to all the subgroup members, for more efficiency). For example, the keys held by the nodes of the Figure 5.3b are summarized in Table 5.1.

Subgroup ID	Node ID	Node key	Pairwise node keys	subgroup key	Pairwise subgroup keys	Group Key
1	1	$K_1^1$	$K_{1,2}^1, K_{1,3}^1$	$K^1$	$K^{1,2}, K^{1,3}$	$K_{(G)}$
	2	$K_2^1$	$K_{2,1}^1, K_{2,3}^1$			
	3	$K_3^1$	$K_{3,1}^1, K_{3,2}^1$			
2	1	$K_1^2$	$K_{1,2}^2, K_{1,3}^2, K_{1,4}^2$	$K^2$	$K^{2,1}, K^{2,3}$	
	2	$K_2^2$	$K_{2,1}^2, K_{2,3}^2, K_{2,4}^2$			
	3	$K_3^2$	$K_{3,1}^2, K_{3,2}^2, K_{3,4}^2$			
	4	$K_4^2$	$K_{4,1}^2, K_{4,2}^2, K_{4,3}^2$			
3	1	$K_1^3$	$K_{1,2}^3$	$K^3$	$K^{3,1}, K^{3,2}$	
	2	$K_2^3$	$K_{2,1}^3$			

Table 5.1 – Example of keys held by nodes.

### 5.3.2 Subgroup Management

The subgroup management consists of distributing nodes on subgroups while minimizing the number of keys they manage. The aim is to improve the protocol efficiency and scalability in heterogeneous networks. In the following, we use the notations  $n$  and  $p$  to refer to the number of nodes and subgroups in the network, respectively. We also denote the number of members of a subgroup  $S$  by  $m^S$ .

A member of  $S$  manages one secret key,  $m^S - 1$  pairwise node keys, one subgroup key,  $p - 1$  pairwise subgroup keys and one group key. The Key Management overhead on nodes is therefore proportional to the sum  $p + m^S$ . Two points come out of this. First, regardless of the subgroup to which a node belongs, the value of  $p$  is the same. Thus, if it is minimized, the overheads are reduced on any node of the group. Moreover, the number of keys held by a node depends on the size of its subgroup. Hence, to balance the loads between the nodes of a heterogeneous network, the most constrained ones must be assigned to the smallest subgroups, and conversely. Indeed, for a node to manage fewer keys than a more powerful one, the former must be assigned to a subgroup smaller than the one to which the latter belongs.

We focus in this section on the management of heterogeneous subgroups, i.e. subgroups of different sizes, while minimizing their number,  $p$ . Note that this does not mean that we do not allow two subgroups to have the same size. To achieve this, we rely on the fact that the nodes of  $S$  must be able to handle at least  $p + m^S$  keys. The size of  $S$  is chosen so that  $p + m^S$  does not exceed the capability of its members or, to put it more simply, the capability  $mc^S$  of its weakest node. Indeed, as  $mc^S$  is the minimum capability that a member of  $S$  can have, if its value is greater than  $p + m^S$  then all the nodes of  $S$  will be able to handle the costs. The problem is, therefore, to choose the minimum capabilities of subgroups and to assign them nodes so as to always satisfy:

$$\min p \tag{5.1}$$

$$\text{under duress: } \forall S, \quad mc^S \geq p + m^S \tag{5.2}$$

On this basis, we propose a heterogeneous subgrouping that takes into account the capabilities of the nodes during their distribution into subgroups. Before we detail this subgroup management, we present the Capability Evaluation Function (CEF) used to evaluate the number of keys a node can manage.

### 5.3.2.1 Capability Evaluation Function

Depending on the application requirements, several parameters can be taken into account to evaluate the number of keys a node can manage. We choose then three types of resources: memory, processing and energy. Indeed, nodes and especially the wireless sensor ones are generally constrained by their limited physical size and so they have limited battery energy supply. Moreover, they have restricted computational capabilities and their memories are limited [53]. We do not consider communication because we will show that, using our solution, its cost is  $O(1)$ .

Therefore, the CEF we propose takes as input the following arguments: the storage capability of a node  $u$  ( $sc_u$ ) and the amount of data that it can process per unit time ( $adt_u$ ) and per unit energy ( $ade_u$ ). Note that the CEF takes into account a percentage of the node resources only to balance the overhead associated to the Key Management against other node requirements. The CEF calculates then  $c_u$ , the number of keys that can be managed by  $u$ . To achieve this, the CEF determines the minimum between the number of keys, of length  $l$ , that the node can store ( $\frac{sc_u}{l}$ ) and the number of keys that it may calculate per unit time ( $\frac{adt_u}{l}$ ) and per unit energy ( $\frac{ade_u}{l}$ ) (Formula 5.3). According to the network and application requirements, weighting can be given to each parameter.

$$c_u = \text{Min}(\frac{sc_u}{l}, \frac{adt_u}{l}, \frac{ade_u}{l}) \quad (5.3)$$

### 5.3.2.2 Heterogeneous subgrouping

The heterogeneous subgrouping management consists of manipulating subgroups of different sizes while minimizing their number and ensuring that the inequality 5.2 is always satisfied. To achieve this, a minimum capability  $mc^S$  is attributed to each subgroup  $S$  when created. To satisfy the inequality 5.2,  $mc^S - p$  nodes are assigned to  $S$  at most ( $m^S \leq mc^S - p$ ). Note that  $mc^S$  must always be greater than  $p$  for  $m^S$  to be greater than 0. Also, the size of a subgroup varies according to its minimum capability and the value of  $p$ . Thus, the greater the capabilities of its members, the larger its size.

A node  $u$ , that can handle  $c_u$  keys, is assigned to  $S$  only if  $mc^S$  is the nearest value less than  $c_u$  ( $mc^S \leq c_u < mc^{S+}$ , while  $mc^{S+}$  is the value that follows  $mc^S$ ). Thus,  $u$  will manage  $p + m^S$  keys, in the worst case. Since the inequality 5.2 is satisfied for  $S$  and  $c_u \geq mc^S$  then  $c_u \geq p + m^S$ . In other words,  $u$  can surely support the overheads. Moreover, thanks to this, the loads are well balanced between the nodes according to their capabilities.



After the assignment, depending on whether  $S$  is an existing subgroup or a new one, the value of  $p$  or  $m^S$  increases. It can happen that for a subgroup  $T$  ( $T$  may be  $S$  or not) the sum  $p + m^T$  exceeds  $mc^T$  and thereby some of its members may not be able to handle all the keys anymore. In this case,  $T$  is splitted into two subgroups having the same minimum capability  $mc^T$ . The size of the resulting subgroups is equal to the half of  $m^T$  and the inequality 5.2 is true again for them. However,  $T$  cannot be splitted if it contains only one node. It is then removed and its member is revoked.

Considering the inequality 5.2 and the fact that  $S$  cannot be empty, any node  $u$  should be able to store at least  $p + 1$  keys (instead of  $\sqrt{n}$  when a homogeneous subgrouping was used). On the other hand, if  $u$  can manage only  $p + 1$  keys then it is the only node of  $S$  and must be revoked when a new subgroup is created. Indeed, if the value of  $p$  increases,  $u$  cannot handle all the keys anymore. For simplicity, we assume that  $u$  is authorized to join the group only if it can store at least  $p$  keys (i.e.  $c_u \geq p$  instead of  $p + 1$ ). Therefore, smaller is  $p$ , the more likely it is that more constrained nodes can join the group. This is one of the reasons why  $p$  should be minimized. For this purpose, depending on the state of the group, subgroups may be merged to reduce their number.

Regarding the choice of the subgroup minimum capabilities, the difficulty lies in the fact that subgroups are created and removed as and when required and that the abilities of nodes are not known a priori. We tried different increasing sequences and found out that the best results (the loads are well balanced and  $p$  is minimized) are obtained when the sequence grows exponentially. Indeed, if the minimum capabilities are close to each other, the subgroups will be well balanced but their number will be too large. However, the aim of the subgrouping is precisely to minimize the number of subgroups and thereby reduce the nodes' storage overhead. We then selected two sequences in particular: powers of two and Fibonacci sequence. Note that other sequences can be used as long as they grow exponentially.

If powers of two are used, the group is partitioned so that a minimum capability is the double of the preceding one (Formula 5.4).

$$mc(l) = \begin{cases} 2.mc(l-1), & \text{if } l > 0. \\ 1, & \text{otherwise.} \end{cases} \quad (5.4)$$

On the other hand, if a Fibonacci sequence is used, a minimum capability is the sum of the two preceding ones (Formula 5.5). Note that  $c_1$  and  $c_2$  are arbitrary constants.

$$mc(l) = \begin{cases} mc(l-1) + mc(l-2), & \text{if } l > 1. \\ c_2, & \text{if } l = 1. \\ c_1, & \text{otherwise.} \end{cases} \quad (5.5)$$

The heterogeneous subgrouping is based on two algorithms: the Assignment and Reorder Algorithms. The Assignment Algorithm is run when nodes join the group and assigns them to the right subgroups. It creates new ones when it is necessary and may split others so that the inequality 5.2 remains always satisfied. The Reorder Algorithm is executed after a node leaving to reduce the number of subgroups. It then removes those that become empty and merges others to the possible extent. Figure 5.5 shows an example of a group partitioned using powers of two. Note that the inequality 5.2 is satisfied for all the subgroups and the value of  $p$  is minimal.

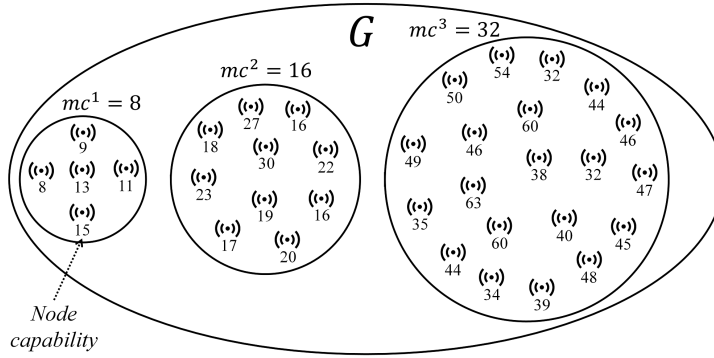


Figure 5.5 – Example of a group partitioned into three subgroups.

### 5.3.2.3 Assignment Algorithm

The Assignment Algorithm is run by the Key Manager when a node  $u$  is authorized to join the group. It takes as input  $c_u$ , the number of keys that can be handled by  $u$ , and assigns it to a subgroup according to the input value. To achieve this, the algorithm manipulates a list of subgroups,  $lsg$ , of size  $p$ . Each of its items contains the *ID* of a subgroup  $S$ ,  $sid^S$ , its minimum capability,  $mc^S$ , and its size,  $m^S$ .

When a node  $u$  is authorized to join the group, the Assignment Algorithm starts by determining the minimum capability  $mc_u$  that matches it. It then rounds down  $c_u$  to the nearest power of two or term of a Fibonacci sequence. Next, it searches in  $lsg$  a subgroup  $S$  such as  $mc^S = mc_u$ . If no subgroup is found (or if the group is empty), a new one is created (See section 4.3.3.1). After that, the algorithm assigns  $u$  to  $S$ , updates  $lsg$  and renews the group security material following the steps described in section 5.3.3.

Also, the algorithm checks if the inequality 5.2 is still satisfied for all subgroups. It browses the list  $lsg$  and as long as there is a subgroup  $T$  for which  $mc^T < p + m^T$ , it is splitted following the steps described in the next paragraph. The size of the resulting subgroups will then be equal to the half of  $m^T$  and the inequality 5.2 will be true again for them. The steps of the Assignment Algorithm are described in Algorithm 3.

---

**Algorithm 3:** Heterogeneous Assignment Algorithm
 

---

**Input** :  $c_u$  = capability of the node  $u$

```

1 Round down  $c_u$  to the nearest minimum capability  $mc_u$ ;
2 Find in  $lsg$  a subgroup  $S$  so that  $mc^S = mc_u$ ;
3 if no subgroup is found then
4   | Create a new one  $S$ ;
5 end
6 Assign  $u$  to  $S$ ;
7 Update  $lsg$ ;
8 while  $\exists T$  for which  $mc^T < p + m^T$  do
9   | Split  $T$ ;
10 end

```

---

To fix ideas, let us consider a node  $u$  ( $c_u \geq 8$ ) which is allowed to join the group  $G$  of the previous example (Figure 5.5). First, the Assignment Algorithm rounds down  $c_u$  to the nearest minimum capability  $mc_u$  and then searches in  $lsg$  a subgroup  $S$  of minimum capacity  $mc^S = mc_u$ . Thus, according to the value of  $c_u$  several cases arise.

- If  $c_u < 16$ , then  $mc_u = 8$  and  $u$  is assigned to  $S^1$ . However, the size of the latter increases and the inequality 5.2 is not true any more for it ( $mc_{S^1} = 8 < p + m_{s^1} = 3 + 6 = 9$ ).  $S^1$  is splitted (Figure 5.6).
- If  $16 \leq c_u < 32$ , then  $mc_u = 16$  and  $u$  is assigned to  $S^2$ . In this case, no split is necessary since the inequality 5.2 is still true for all the subgroups.
- If  $32 \leq c_u < 64$ , then  $mc_u = 32$  and the node  $u$  is assigned to  $S^3$ . Also, no split is necessary since the inequality 5.2 is still true for all the subgroups.
- If  $c_u \geq 64$ , then  $mc_u \geq 64$ . As there is no subgroup having a minimum capacity greater than 64, a new one ( $S^4$  with  $mc^4 = mc_u$ ) is created and the node  $u$  is assigned to it. However, by creating a new subgroup, the value of  $p$  increases. Thus, the inequality 5.2 is not true any more for  $S^1$  ( $mc_{S^1} = 8 < p + m_{s^1} = 4 + 5 = 9$ ).  $S^1$  is splitted.

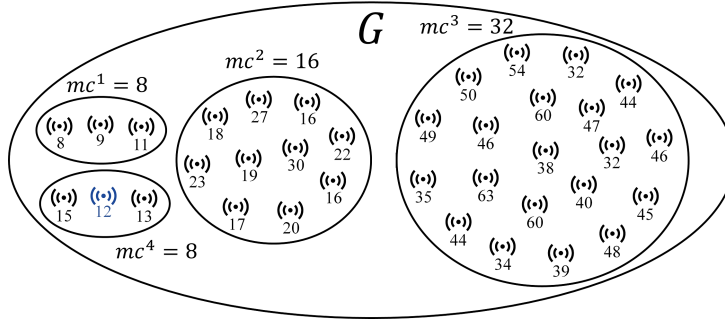


Figure 5.6 – Example of a subgroup splitting.

**Subgroup splitting:** Splitting a subgroup  $S$  consists first of creating a new subgroup  $T$  ( $mc^T = mc^S$ ). The  $\frac{mc^S}{2}$  last nodes that have joined  $S$  are then moved to  $T$ . We denote by  $S^+$  the subgroup  $S$  after being splitted and by  $u_f$  the first node of  $S$  to join  $T$ , i.e.  $\forall u \in S^+, nid_u < nid_f$  and  $\forall v \in T, nid_v \geq nid_f$ .

The Key Manager starts by determining  $sid^T$ . Next, to ensure the forward secrecy, it randomly generates two refresh keys,  $K_{R_1}$  and  $K_{R_2}$ . Then, using the  $KDF$ , it computes  $K^{S^+}$  and  $K^T$  (Formulas 5.6 and 5.7). After that, all the pairwise keys associated to two nodes that no longer belong to the same subgroup are removed. Also, for each subgroup  $U$  (including  $S$ ), a pairwise subgroup key  $K^{T,U}$  is created.

$$K^{S^+} = KDF(K^S || K_{R_1}) \quad (5.6)$$

$$K^T = KDF(K^S || K_{R_2}) \quad (5.7)$$

Furthermore, the algorithm sends the unicast message  $SM1$  to each node  $u \in S^+$  ( $nid_u < nid_f$ ). The message is encrypted by means of the node secret key and contains  $K_{R_1}$  as well as the pairwise subgroup key  $K^{S,T}$ . It also sends the unicast message  $SM2$  to each node  $v \in T$  ( $nid_v \geq nid_f$ ) encrypted using the node secret key.  $SM2$  contains  $K_{R_2}$  and  $L_v$ , the list of the new generated pairwise subgroup keys that are associated to  $T$ . Finally, the unicast message  $SM3$  is sent to each subgroup  $U$  ( $U \neq S$  and  $U \neq T$ ). It is encrypted by means of the node secret key and contains the pairwise subgroup key  $K^{T,U}$ .

$$SM1 : KM \rightarrow u : < \{nid_f, K_{R_1}, K^{S,T}\} K_u^S >$$

$$SM2 : KM \rightarrow v : < \{nid_f, K_{R_2}, L_v\} K_v^T >$$

$$SM3 : KM \rightarrow U : < \{K^{T,U}\} K^U >$$

### 5.3.2.4 Reorder Algorithm

The Reorder Algorithm is run, after a node leaving (the node can voluntarily leave the group or be evicted because it gets compromised), to reduce the number of subgroups ( $p$ ). The aim is to decrease the number of keys that the group members has to handle. Thus, the algorithm takes as input the percentage of merging,  $pcm$ , and tries to remove or merge subgroups when it is possible. To achieve this, the algorithm manipulates the same list  $lsg$  used by the Assignment Algorithm.

When a node leaves a subgroup  $S$ , the algorithm checks the number of the remaining ones. If  $S$  becomes empty, it is removed (See section 4.3.3.2). If the size of  $S$  falls below a certain threshold,  $thr$ , the algorithm searches in  $lsg$  a subgroup  $T$  to merge with  $S$ . The threshold is the product of the percentage of merging and the maximum size of  $S$  ( $thr = pcm.(mc^S - p)$ ). Furthermore,  $T$  must have the same minimum capability as  $S$  and its current size must also be less than the threshold. If it is the case, the two subgroups are merged following the steps described in the next paragraph. Finally, the Key Manager renews the network security material, following the steps described in the node management section (See section 5.3.3). The steps of the Reorder Algorithm are described in algorithm 4.

Note that  $pcm$  must not exceed 50% so that the size of the resulting subgroup does not exceed  $mc^S - p$ . Also, the greater is  $pcm$ , the more the subgroups are merged. This increases the merging's cost but reduces the value of  $p$ . The value of  $pcm$  defines then a compromise between the merging's overheads and the value of  $p$ .

**Subgroup merging:** Merging  $S$  and  $T$  consists of three steps. A new subgroup  $S^+$  is first created (See section 4.3.3.1). Next, the members of  $S$  and  $T$  are moved to the new subgroup. New pairwise keys are then generated for every pair of nodes  $u, v$  ( $u \in S$  and  $v \in T$ ) and sent to them (Messages  $MM1$  and  $MM2$ ). These messages are encrypted by means of the node keys. They contain the new cognate  $ID$  and the pairwise key associated to it. The message  $MM2$  also includes the list ( $L_v$ ) of the pairwise subgroup keys related to  $S$ . Finally, the two subgroups  $S$  and  $T$  are removed.

$$MM1 : KM \rightarrow u : < \left\{ nid_u, K_{u,v}^{S^+} \right\}_{K_u^{S^+}} > (\forall u \in S, \forall v \in T)$$

$$MM2 : KM \rightarrow v : < \left\{ nid_v, K_{u,v}^{S^+}, L_v \right\}_{K_v^{S^+}} > (\forall v \in T, \forall u \in S)$$

**Algorithm 4:** Heterogeneous Reorder Algorithm

---

**Input** :  $pcm$  = percentage of merging

```

1 foreach subgroup  $S$  that a node has left do
2   if  $m^S = 0$  then
3     | Remove  $S$ ;
4   end
5   else
6     |  $thr \leftarrow pcm.(mc^S - p)$ ;
7     | if  $m^S < thr$  then
8       | Find in  $lsg$  a subgroup  $T$  such as  $m^T < thr$  and  $mc^T = mc^S$ ;
9       | if a subgroup is found then
10      | Merge  $S$  and  $T$ ;
11      | end
12    | end
13  end
14 end

```

---

**5.3.3 Node management**

The way the keys are managed upon a network change is almost similar to that of the version intended for device-to-device communication (See section 4.3.4). The main difference with this new solution is the management of the group key. Thus, when a node joins or leaves the network, this key must be replaced in the same way as the subgroup and pairwise subgroup keys. The aim is to ensure the backward and forward secrecy. Thus, the Key Manager uses the  $KDF$  and a randomly generated refresh key to update the group key (Formula 5.8).

$$K_{(G)}^+ = KDF(K_{(G)} || K_R) \quad (5.8)$$

So that the group members can update the group key, they must know the refresh key used by the Key Manager. It is worth repeating that the message  $JM2$  containing the refresh key is broadcast to each subgroup (See section 4.3.4). When a group member receives the message, it first decrypts it using its subgroup key. Next, it uses  $K_R$  and the  $KDF$  to update the group key (Formula 5.8). After that, the node discards the refresh key. In the case of a node joining, the Key Manager agrees with  $u$  on a temporary secret key (using a key agreement method). This key is then used to securely provide the joining node with the group key.

## 5.4 Security analysis

In this section, we analyze the security of our solution. We then prove that it guarantees the backward and forward secrecy. We also show that it resists to collusion attacks.

### 5.4.1 Backward secrecy

We prove that a joining node cannot access the current group key or any previous incarnation of it. The same goes for the subgroup and pairwise subgroup keys related to its subgroup.

**Proposition 5.1:** Backward secrecy is guaranteed as a joining node never gets knowledge of the old security material used before it joins the group.

**Proof:** Let us consider a node  $u$  that joins a subgroup  $S$ . The Key Manager starts by updating the keys mentioned above. Then, before  $u$  can actually join the group, the Key Manager rekeys all current members of the network, by means of messages  $JM1$  and  $JM2$ . These messages are encrypted by means of their node and subgroup keys, respectively. Since none of these keys are known to  $u$ , the joining node is excluded from the process of rekeying.

### 5.4.2 Forward secrecy

We prove that a leaving node cannot access the new group key or any future incarnation of it. The same goes for the subgroup and pairwise subgroup keys associated to its subgroup.

**Proposition 5.2:** Our solution guarantees the forward secrecy since a leaving node does not have access to the new security material.

**Proof:** Let us consider a node  $u$  that leaves a subgroup  $S$ . The Key Manager rekeys the subgroup members and the rest of the nodes by means of the messages  $LM1$  and  $LM2$ , respectively. The former is encrypted by means of the node keys and the latter using the subgroup keys. Since none of these keys are known to  $u$ , the leaving node is excluded from the process of rekeying.

### 5.4.3 Collusion resistance

We prove that our solution is resistant to collusion attacks.

**Proposition 5.3:** Our solution resists to collusion attacks as multiple evicted nodes cannot cooperate to regain access to the network.

**Proof:** When nodes are evicted because they get compromised, the Key Manager rekeys the group members by means of the messages  $LM1$  and  $LM2$ . The former is encrypted by means of the node keys and the latter using the subgroup keys. Since these keys are independent of each other and as none of them are known by the evicted nodes, these nodes cannot collude to decipher the rekeying messages.

## 5.5 Performance evaluation

In this section, we evaluate the performance of our solution. After showing that it resists to collusion attacks and that it guarantees the backward and forward secrecy, we prove that, by balancing the loads between the heterogeneous devices according to their capabilities, our solution is both efficient and highly scalable.

### 5.5.1 Theoretical analysis

We begin by briefly analyzing the overheads of our solution on the Key Manager before detailing them on the network members.

#### 5.5.1.1 Overheads on the Key Manager

We start by analyzing the overheads of our solution on the Key Manager side.

**Property 5.1:** The communication overhead of an operation related to a subgroup  $S$  is proportional to the sum  $p + m^S$  on the Key Manager.

**Proof:** Regardless of the rekeying operation performed (such as a node joining  $S$  or a node leaving  $S$ ), the Key Manager sends a unicast message to each of the  $m^S$  members of  $S$  and broadcasts a message for each of the other  $p - 1$  subgroups, in the worst case. The Key Manager then sends a total number of messages proportional to the sum  $p + m^S$ .



---

**Property 5.2:** The calculation overhead of an operation related to a subgroup  $S$  is proportional to the sum  $p + m^S$  on the Key Manager.

**Proof:** Regardless of the rekeying operation performed (such as a node joining  $S$  or a node leaving  $S$ ), the Key Manager updates the keys which are or will be known by the node in question. The calculation overhead on the Key Manager is therefore proportional to the storage cost on nodes, which will be proven to be of the order of  $p + m^S$  in the next Section. Also, the subgroup management algorithms browse the list of subgroups in the worst case. They therefore have a complexity proportional to  $p$ .

**Property 5.3:** The storage overheads on the Key Manager is proportional to  $O(n)$ .

**Proof:** The number of nodes is more important than that of subgroups. If we choose not to store the pairwise node keys (used to secure communication between the nodes) in the Key Manager's memory, the largest number of keys to store will be that of the node secret keys. The Key Manager will then store a number of keys proportional to  $n$ .

**Discussion:** The communication and calculation costs of an operation related to the subgroup  $S$  are proportional to  $p + m^S$  on the Key Manager. The storage, on the other hand, is of the order of  $O(n)$ . Considering the significant improvement (presented in the next section) that our solution provides on the node side, the costs on the Key Manager are reasonable. Also, as the Key Manager has usually plentiful of resources, we aimed to make the costs more affordable on the nodes, especially the constrained ones.

### 5.5.1.2 Overheads on nodes

Now, we analyze the overheads of our solution on the node side.

**Property 5.4:** The communication overhead on the nodes is  $O(1)$ .

**Proof:** Regardless of the rekeying operation performed (such as a node joining  $S$  or a node leaving  $S$ ), a node receives a constant number of messages. The communication cost on nodes is therefore independent of the network size.

**Property 5.5:** The calculation overhead of an operation related to a subgroup  $S$  is proportional to the sum  $p + m^S$  on the nodes.

**Proof:** Regardless of the rekeying operation performed (such as a node joining  $S$  or a node leaving  $S$ ), a node updates all the keys it knows, in the worst case. The calculation cost on the nodes is therefore proportional to the storage, which will be proven to be proportional to the sum  $p + m^S$  in the next proof.

**Property 5.6:** The storage overhead on a member of a subgroup  $S$  is proportional to the sum  $p + m^S$ .

**Proof:** Using our protocol, a node of a subgroup  $S$  stores one secret key,  $m^S - 1$  pairwise node keys, one subgroup key,  $p - 1$  pairwise subgroup keys and a group key. The storage overhead on the node is therefore proportional to the sum  $p + m^S$ .

**Discussion:** The communication cost of our solution on the members of a subgroup  $S$  is  $O(1)$ , while the calculation and storage overheads are proportional to the sum  $p + m^S$ . Thus, to reduce these costs as well as those of the Key Manager, we aimed to minimize the number of subgroups,  $p$ . We then implemented a simulator to analyze the behaviour of its value according to several parameters.

### 5.5.2 Simulation

The simulator, we implemented in C, randomly generates node capabilities (based on a uniform distribution) and runs the Assignment Algorithm to distribute them into subgroups. It also simulates nodes leaving and runs the Reorder Algorithm. The simulator takes as inputs the following parameters:

- **n** : the number of nodes (Default value: *1024000*);
- **C\_MAX** : the maximum value that the simulator can generate (Default value: *256000*);
- **pcm** : the percentage of merging (Default value: *0.3*);
- **t** : the subgrouping type (Default value: powers of two).

The simulator then outputs the number of subgroups, which is represented by the size of the list *lsg*. This allows us to analyze the effect of the above-mentioned parameters on the value of  $p$ . Each time we set three parameters to default values and we vary the fourth.

**Number of nodes:** Starting with the network size, the results of the simulations are plotted in Figure 5.7. They show that, regardless of the network size, by using our method of load balancing the number of subgroups is reasonable. Figure 5.7 shows that even when the size of the network exceeds one million of nodes, the value of  $p$  does not exceed a few dozen. This makes our solution scalable since the constrained nodes manipulate a reasonable number of keys.

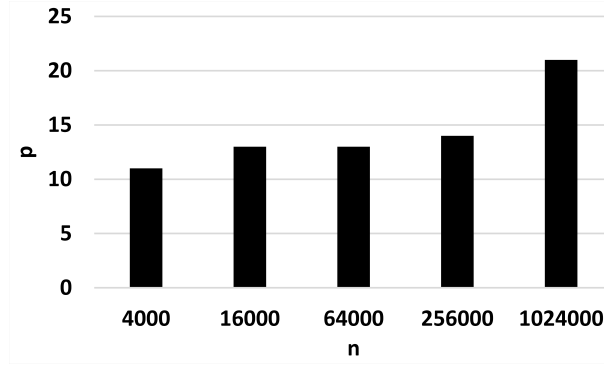


Figure 5.7 – Effect of the number of nodes on the value of  $p$ .

**Maximum capability of nodes:** Now, we analyze the effect of the maximum capability that can be generated by the simulator. The results are plotted in Figure 5.8. They show that the more powerful the nodes are, the smaller the value of  $p$  is. This is because powerful devices are able to manage more keys and can be assigned to larger subgroups. Note that the larger the subgroups are, the more their number diminishes. Therefore, since the costs of our solution on the constrained nodes mainly depend on the number of subgroups, they are more likely to support the overheads if the network becomes too large. Moreover, even when the maximum capability is small, the value of  $p$  remains reasonable for a network containing over a million nodes. To sum up, our solution is scalable regardless of the nodes maximum capabilities but it can become even more if the network contains enough powerful members.

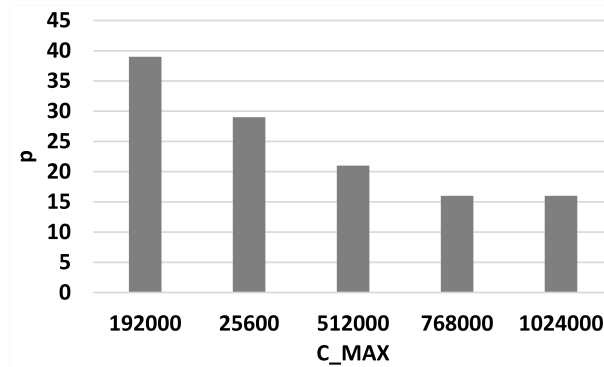


Figure 5.8 – Effect of the maximum capability of nodes on the value of  $p$ .

**Percentage of merging:** Now, we study the effect of the percentage of merging. The results of the simulations are plotted in Figure 5.9. They show that the greater the percentage of merging is, the smaller the value of  $p$  is. Therefore, the merging operation actually reduces the number of subgroups and makes our solution lighter for the constrained devices and thereby more scalable. Note that most of the overheads imposed by the subgroup merging are at the level of the Key Manager and have no significant influence on the performance of nodes.

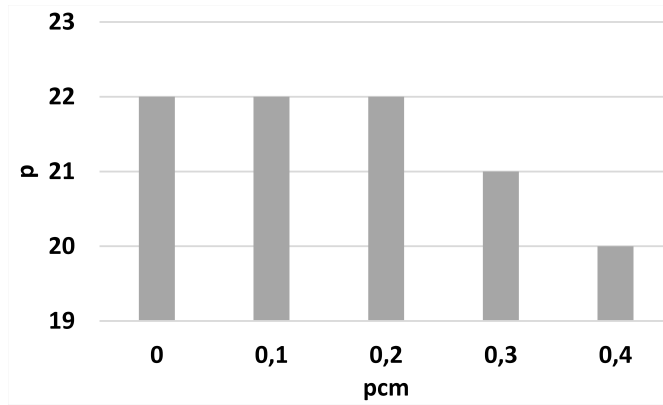


Figure 5.9 – Effect of the percentage of merging on the value of  $p$ .

**Subgrouping type:** Finally, we study the effect of the type of subgrouping. The results of the simulations are plotted in Figure 5.10. They show that the use of powers of two gives slightly better results than a Fibonacci sequence. However, they generally gives approximately the same results for large networks such as the IoT.

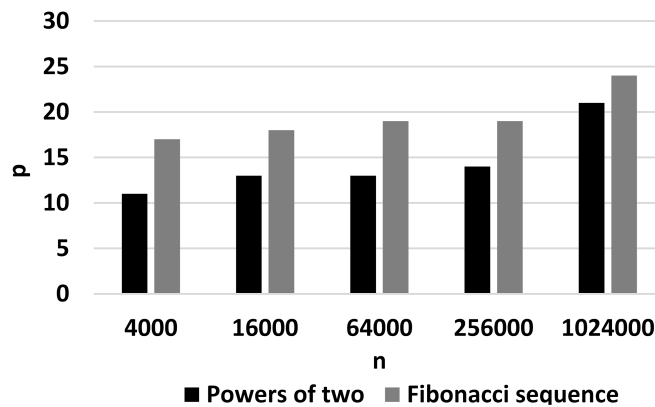


Figure 5.10 – Effect of the subgrouping type on the value of  $p$ .

### 5.5.3 Comparison

After showing that our solution resists to collusion attacks and guarantees the backward and forward secrecy, let us prove that it meets the other IoT requirements: efficiency and scalability. Thus, we compare our solution to the version presented in the previous chapter (DKM) as well as to the Key Management protocol presented in [129] (GREP) to secure group communication.

### 5.5.3.1 Efficiency and scalability

The communication cost of our solution on nodes is  $O(1)$  and therefore does not need to be discussed anymore. On the other hand, the calculation cost on nodes is proportional to storage. For these reasons, we only need to analyze the storage costs to compare the efficiency and scalability of our solution to those of the existing schemes. We then take as example a TmoteSky sensor node and consider keys of 256 bits (using AES-256 for example). Featuring 48 Kbytes, a TmoteSky can store up to 1536 keys (ignoring the other node's memory requirements).

For the node to support the storage cost of our solution, it is enough if it can store at least  $p$  keys. The percentage of storage capability to indicate to the CEF must then be greater or equal to  $P_o = \frac{p}{1536}$ . On the other hand, using GREP or DKM the storage cost is proportional to  $O(\sqrt{n})$ . The memory rate required to store these  $\sqrt{n}$  keys is therefore  $P_r = \frac{\sqrt{n}}{1536}$ . We compare the variation of the two values according to  $n$ . To achieve this, we used the default values of the algorithms' parameters. The results of the simulations are plotted in Figure 5.11.

It is important to highlight that our solution requires less storage on a TmoteSky than the other protocols. Indeed, the value of  $P_o$  is smaller than  $P_r$ , no matter the group size. More importantly, if the group contains one million nodes, more than half of the memory of the TmoteSky will be used to store all the keys using GREP or DKM. On the other hand, under the conditions of the simulations, less than 2% of its storage capability is enough if a our solution is used. This is because storage cost is well balanced between the group members according to their capabilities. Thus, by using a bit more of the resources of powerful devices, our solution becomes much lighter for the constrained ones. It can then operate on much larger heterogeneous networks such as the IoT.

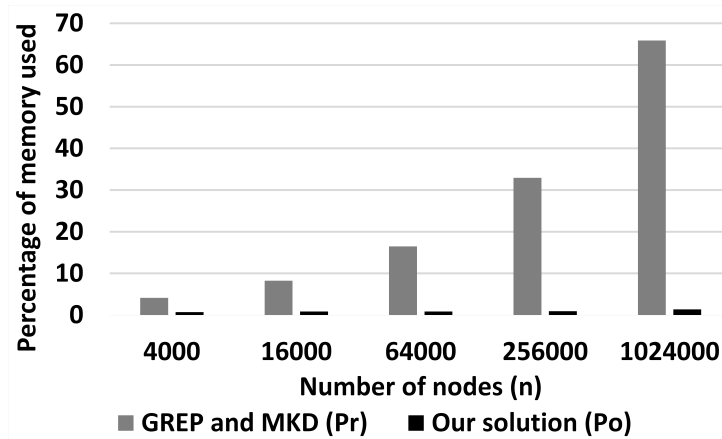


Figure 5.11 – Efficiency and scalability.

### 5.5.3.2 Heterogeneity

Unlike most of the existing protocols, our solution balances the loads between the heterogeneous devices of the network according to their capabilities. To illustrate this difference, we consider the protocols DKM and GREP, which have storage and calculation cost proportional to  $O(\sqrt{n})$ . Note that these costs are the same for all the group members, while they are both proportional to the nodes' storage capability, using our solution. We consider then a network of 10000 nodes and analyze the variation of the calculation and storage cost according to the node's storage capability (number of keys it can store), for the three protocols. Note that the percentage of storage capability that we choose to indicate to the CEF is 10% (i.e. only 10% of the real capability of the node is used). The results are plotted in Figures 5.12.

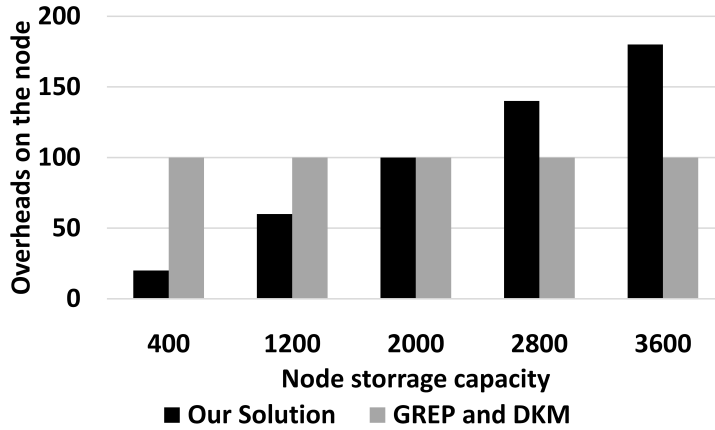


Figure 5.12 – Heterogeneity.

We take as example two nodes  $u_1$  and  $u_2$  that can store 200 and 1800 keys, respectively. For both nodes, 10% of their memory is used by our solution, in the worst case. DKM and GREP, on the other hand, use 50% of the former and 5% of the latter. As the calculation overhead on node depends on the storage, these protocols quickly exhausts the resources of  $u_1$ , while  $u_2$  has much more. More importantly, the nodes having a capability lower than 100 can not even store all the keys, while our solution uses 10% of their memory only. Thus, although the overheads imposed by DKM and GREP are lower than that of our solution for powerful devices (capability greater than 1000), they are much greater for the weak ones.

---

## 5.6 Conclusion

The purpose of this chapter was to present a novel versatile Key Management protocol for the IoT. In addition to securing both device-to-device and group communication, our solution considers the heterogeneous nature of the IoT. It is then lighter for the constrained devices by using a bit more of the resources of the powerful ones. Thus, the network performance is significantly improved and its lifetime increased. We showed that our solution ensures the forward and backward secrecy, as nodes can securely join and leave the network at any time. We also proved that our protocol resists to collusion attacks, since multiple evicted nodes can not cooperate to regain access to the network. We finally showed that our solution is both efficient and highly scalable by balancing the loads between the heterogeneous devices according to their capabilities.

In the next chapter, we will present our third contribution. It is a novel decentralized blockchain-based Key Management protocol for secure device-to-device, group and multi-group communication in the IoT. This new solution belongs to the class  $\mathcal{KM}_{post}^{hyb,het}$ .





# *Blockchain-Based Decentralized Key Management for secure Multi-group Communication*

---

In this chapter, we propose a novel decentralized blockchain-based Key Management protocol for the IoT. In addition to securing the device-to-device and group communication, this new solution considers the multi-group communication. It then guarantees the secure coexistence of several services in a single network. To achieve this, our solution manages several groups with independent security parameters. To decentralize the Key Management, we use the blockchain technology and smart contracts. We show that our solution solves the single point of failure problem, since the system continues to operate when a Key Management entity fails. We also prove that resilience is improved, as the compromise of one of these entities does not jeopardize the whole network.

## 6.1 Introduction

In multi-group communication, a device sends/receives multicast messages only to/from the members of one of the groups to which it belongs [62]. Unlike group communication, in which a device is assumed to be part of a single group, this mode considers the possibility that devices can participate in multiple services at the same time. The IoT is indeed a network that can be simultaneously shared by different services. Smart ambulances [113] are an example of technologies for which the multi-group communication is necessary. They can indeed participate, at the same time, in the intelligent transportation network and the healthcare system. The multi-group communication requires that the Key Management guarantees independence of services, efficiency and scalability.

Most of the existing Key Management protocols, especially those designed for group communication, use the same security parameters to secure all the network communications. Thus, if several services are provided by the network, communications within a service will be accessible to all the network members even those which did not subscribe to it. The compromise of a node will then jeopardize all network services. Furthermore, the existing solutions, based on post-distribution, generally require a third party to manage the keys. If this entity is centralized, it will become a single point of failure and the main target of attacks. When it fails, the entire system will stop operating, and if it is attacked, the whole network will be compromised.

To address these issues, we propose a novel decentralized blockchain-based Key Management protocol for the IoT. This new solution belongs to the class  ${}^{hyb}_{hyb}\mathcal{KM}^{het}_{post}$ . We then improve our previous protocol so that it secures, in addition to the device-to-device and group communication, the multi-group communication. It therefore guarantees the secure coexistence of several services in a single network. To achieve this, our solution manages several groups with independent security parameters so that the compromise of a service has no effect on the others. Furthermore, we use the blockchain technology and smart contracts to decentralize the Key Management. We design a lightweight consensus algorithm that takes into account the capability of the blockchain participants for block validation. We show that this solves the single point of failure problem, since the system continues to operate when an entity fails. We also prove that the resilience of our solution is improved, as the compromise of an entity does not jeopardize the whole network.

The remainder of this chapter is organized as follows. Section 6.2 presents related works. Section 6.3 introduces our solution. Section 6.4 presents the security analysis. Section 6.5 introduces the performance evaluation. Section 6.6 concludes the chapter.

## 6.2 Related Works

In this chapter, we are interested in the device-to-device, group and multi-group communication. However, as the first two were detailed in the previous chapters, we focus in this section on the latter. We also discuss the post-distribution Key Management solutions based on the blockchain technology. We then consider the class  ${}^{net}_{*}\mathcal{KM}^{post}_{*}$ .

### 6.2.1 Multi-Group Key Management schemes

As far as we know, only few researches considered the possibility of coexistence of several services in a single network. The authors in [123] and [147] proposed Key Management schemes for hierarchical group access control. However, the protocol cannot achieve a high performance when no hierarchy exists among services. The authors in [99] proposed then a new scheme called the Master Key Encryption Based Key Management. This protocol is nevertheless based on asymmetric cryptography as it was proposed for traditional Internet. It is therefore not well suited for networks containing highly constraint devices such as the IoT.

### 6.2.2 Blockchain solutions

A blockchain is a decentralized and secure storage technology. It relies on cryptography, smart contracts and consensus algorithms to securely replicate an application on several entities. For more details, please refer to section 2.4.

The term blockchain first appeared in Nakamoto's Bitcoin paper describing a new decentralized cryptocurrency [93]. The technology started then to be used in various applications. Recently, researchers began to take interest in using it to decentralize the Key Management. The authors of [78, 79] proposed a blockchain-based Key Management system to secure the group communication in intelligent transportation systems. In [84], a blockchain was used to decentralize the Key Management for hierarchical access control in the IoT. These works do not consider the device-to-device or the multi-group communication and use the Proof of Work (PoW) [93] consensus algorithm. Our solution secures the three modes of IoT communication: device-to-device, group and multi-group communication. It is also based on a version of Proof of Stake (PoS) [132] that takes into account the capability of the blockchain participants. More importantly, PoS is known to be far less energy-intensive than PoW [112].

## 6.3 Our solution

Our literature review shows that none of the existing solutions, proposed for secure multi-group communication or based on the blockchain technology, is well suitable for the IoT. We then propose a novel Key Management protocol [67, 71] belonging to the class  $\mathcal{KM}_{hyb}^{hyb} \mathcal{M}_{post}^{het}$ . Our solution is organized into two layers (Figure 6.1).

The first layer manages the nodes and their cryptographic keys. As the subgroup management was detailed in the previous chapters, we focus in this section on the group management. It is about handling multiple groups and assigning nodes to them according to the services to which they subscribe. The network is then divided into several groups, each of which is also partitioned into several subgroups (Figure 6.2b). By doing this, the security parameters of services will be independent so that the compromise of one of them will have no effect on the others. Moreover, as shown in the previous chapters, the subgrouping makes the protocol lighter for nodes, especially the constrained ones.

The blockchain layer manages the blockchain and its participants. The aim is to securely decentralize the Key Management. It then guarantees that the system continues to operate even if some of its participants fail or are the target of malicious attacks. It also ensures that the compromise of a participant does not jeopardize the security of the entire network.

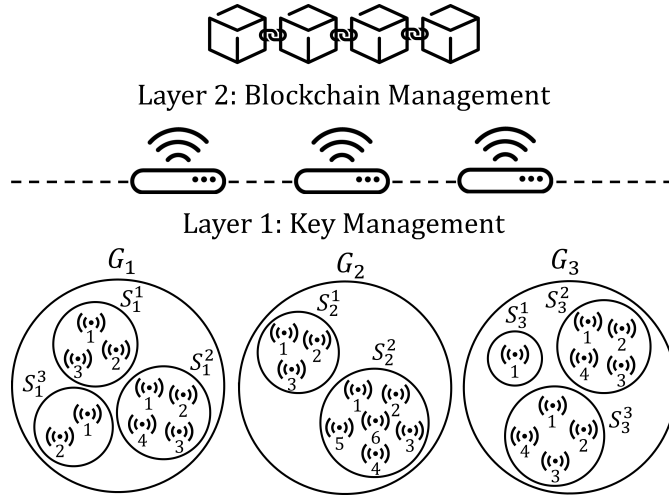


Figure 6.1 – Architecture of our solution.

Our solution is a hybridization of symmetric and asymmetric cryptography. The aim is to take advantages of each and overcome its disadvantages. Symmetric encryption is mainly used in layer 1, while asymmetric encryption is only used in layer 2. Note that the blockchain is implemented on powerful servers (the cloud) or IoT gateways and is completely separated from the constrained devices. The goal is to not involve any additional cost on them, except for those imposed by the first layer.

### 6.3.1 Layer 1: Key Management

The first layer manages the nodes. It organizes them into groups (according to the services in which they participate) and subgroups (according to their capabilities). It also provides them with the keys and the secret materials that allow them to update these keys.

#### 6.3.1.1 Group and service management

An IoT service is a transaction between two entities: a provider and a consumer. The former measures the state of the latter or initiates actions which will cause a change to it [128]. A device may participate (as a provider, a consumer or both) to different services, at the same time, and subscribe or unsubscribe from services at any time. The IoT can then be seen as a set of overlapping classes each gathering nodes that collaborate to provide a service and others that benefit from it (Figure 6.2a). As these classes are overlapping, a group of the protocol cannot be associated to a service. Indeed, the independence of the group security parameters will lose its meaning and the compromise of a node can jeopardize several groups. We propose then the creation of a group for each possible combination of services. A combination  $A$  of  $k$  services, of a finite set  $F$  of  $e$  services, is a subset of  $k$  ( $k < e$ ) elements of  $F$ . The number of possible combinations,  $nc$ , is equal to:

$$nc = \sum_{k=1}^e C_e^k = 2^e - 1 \quad (6.1)$$

The network  $N$  is partitioned into groups. Each group  $G$  is associated with an  $ID$ ,  $gid_{(G)}$ , which is unique within  $N$ . It contains the nodes participating in the services of the combination  $A_{(G)}$  associated to it. When an actual member subscribes or unsubscribes from services, it migrates from a group to another according to its new combination of services. The number of groups can reach  $nc$  (Formula 6.1) if there are nodes participating in every possible combination of services. However, it cannot exceed the number of nodes,  $n$ , as empty groups are not allowed. The maximum number of groups,  $max_g$ , is equal to:

$$max_g = \text{Min}(2^e - 1, n) \quad (6.2)$$

The probability that each node belongs to a distinct group is very low or even impossible in reality. We then assume, in the rest of the chapter, that the  $n$  nodes of the network are uniformly distributed in  $\sqrt{n}$  groups of  $\sqrt{n}$  members each. This is closer to reality, on the one hand, and simplifies the evaluation of our solution's overheads, on the other.

In Figure 6.2a, two services  $E^{(1)}$  and  $E^{(2)}$  coexist in a network  $N$ . Three combinations are then possible:  $A_{(1)} = \{E^{(1)}, E^{(2)}\}$ ,  $A_{(2)} = \{E^{(1)}\}$  and  $A_{(3)} = \{E^{(2)}\}$ . Each group  $G$  contains the nodes participating in the combination of services  $A_{(G)}$  associated to it. The group  $G_1$  contains the nodes participating in both services while those of  $G_2$  and  $G_3$  subscribed only to  $E^{(1)}$  and  $E^{(2)}$ , respectively (Figure 6.2b).

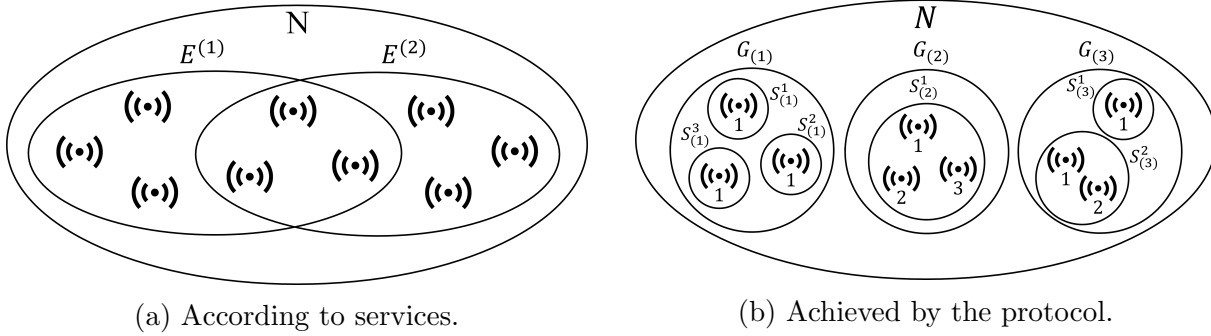


Figure 6.2 – Network partitioning.

### 6.3.1.2 Classification of cryptographic keys

The keys presented in the previous chapters (See Section 4.3.1) are also used by this new solution. However, each of them, especially the *TEKs*, are specific to a single group and independent from one group to another. There are mainly three new keys holds by a node that belongs to a group  $G$ :

- A service key,  $K^{(E)}$ , for each service  $E$  in which the node participates. This is a *DEK* that replaces the group key presented in Section 4.3.1. It is used to secure the group communication between the service members.
- A pairwise service key,  $K_{(G,I)}$ , for each group  $I$  containing members that participate in the same service as those of  $G$  (i.e.  $A_{(G)} \cap A_{(I)} \neq \emptyset$ ). It is a *DEK* used to secure the device-to-device communication between the members of the two groups ( $G$  and  $I$ ).
- A group Key,  $K_G$ . This is a *TEK* used to secure the communication with the Key Manager. It replaces the node and subgroup keys when the same message is sent to all the group members (for more efficiency). It is known only by the members of  $G$ .

### 6.3.1.3 Node Management

Like the subgroup and pairwise subgroup keys, those mentioned above must be updated upon a change in the network (a node  $u$  joins or leaves a group  $G$ ). The aim is to guarantee the backward and forward secrecy. To achieve this, the Key Manager uses the same refresh key and key derivation function used before (Formulas 6.3, 6.4 and 6.5).

$$K^{(E)+} = KDF(K^{(E)} || K_R) \quad (6.3)$$

$$K_{(G,I)}^+ = KDF(K_{(G,I)} || K_R) \quad (6.4)$$

$$K_{(G)}^+ = KDF(K_{(G)} || K_R) \quad (6.5)$$

After the key updates is complete, the Key Manager must rekey the nodes. It then sends the messages  $JM1$  and  $JM2$  to rekey the members of the group  $G$  (See Section 4.3.4). It also multicasts the message  $RM$  (the same message whether its a join or leave operation), encrypted using  $K_{(I)}$  to all the members of the group  $I$  ( $A^{(G)} \cap A^{(I)} \neq \emptyset$ ). The message contains the  $ID$  of the group  $G$  and the refresh key allowing the members of  $I$  to update their pairwise service key  $K_{(G,I)}$ .

$$RM : KM \rightarrow I : < \{gid_{(G)}, K_R\}_{K_{(I)}}, \forall I \in N, \text{ such that } A^{(G)} \cap A^{(I)} \neq \emptyset$$

When a member of  $I$  receives the message  $RM$ , it first decrypts it, using its group key, and retrieves its contents. The node then uses the  $KDF$  and  $K_R$  to update the pairwise service key,  $K_{(G,I)}$ , it shares with the members of  $G$ .

### 6.3.1.4 Subgroup management

The subgroup management remains almost the same as in the previous chapters. It consists of distributing nodes on subgroups uniformly (If a homogeneous subgrouping is used, see section 4.3.3) or according to their capabilities (If a heterogeneous subgrouping is used, see section 5.3.2). The aim is to improve the protocol efficiency and scalability. The only difference in this new solution is that the subgroups of each group are managed independently of each other.

### 6.3.2 Layer 2: Blockchain Management

The purpose of this layer is to decentralize the Key Management using the blockchain technology and smart contracts. Although any type of architecture can be used (since the secret keys are not stored in the blockchain), a private or a consortium blockchain remains preferable in an application such as the Key Management. A limited number of participants also makes the blockchain management more efficient. For more details about the blockchain architectures, please refer to Section 2.4.2. We introduce into the network IoT gateways (or *BPs* for Blockchain Participants) that generate, validate and store transactions upon a network change. The *BPs* can also be implemented on servers.

The *BPs* act as intermediaries between the nodes and the blockchain (Figure 6.1). The aim is to not involve any additional cost on nodes, except those imposed by Layer 1. When a node wishes to join the network, it sends a request to a *BP*. If the transaction corresponding to this request is validated by the other *BPs* and is correctly added to the blockchain, the node is attached to the gateway that initiates the joining process. It will remain attached to it until the node moves, leaves the network or when the *BP* fails or gets compromised (see section 6.3.2.3). Meanwhile, the *BP* manages (generates, stores and updates) the keys associated to the node. The *BP* also sends to the node the rekeying messages, so that it can update its keys.

A blockchain transaction is the storage unit that corresponds to a specific event, which is a rekeying operation in our case. Any *BP* that executes in order the operations stored in the blockchain should have the same organization of nodes in groups and subgroups. As shown in Figure 6.3, a transaction contains the following information: the rekeying operation (join, leave or evict), the node ID, the node capability, the ID of the subgroup of the node, the ID of the group of the node, the cryptographic hash of the node’s secret code (see section 6.3.2.3) and the refresh key used to update the keys.

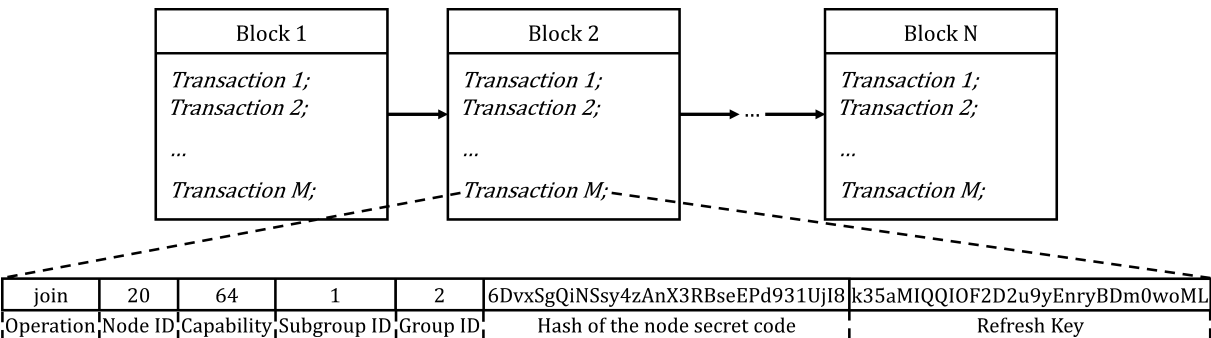


Figure 6.3 – Example of a blockchain transaction.



### 6.3.2.1 Transaction management upon network change

When a *BP* receives a join or leave request, it first uses the layer 1 for the group, subgroup and node management. The algorithms and operations performed by this layer (e.g. Assignment and Reorder Algorithms) are transformed into smart contracts. The aim is to ensure that, for a certain join or leave request, any *BP* should obtain the same result. Before distributing the keys, the layer 1 calls the layer 2 to generate, validate and store a transaction in the blockchain. If the transaction corresponding to the current rekeying operation is correctly stored in the blockchain, the layer 2 informs the layer 1. The *BPs* can then distribute the generated keys on the appropriate nodes after ciphering them using the *KEKs* (Figure 6.4).

**Transaction generation:** When the layer 2 receives the information from the first layer about a rekeying operation, it starts by generating the corresponding transaction. The layer 2 then stores it in its memory pool of temporary transactions and broadcasts it to all the *BPs*. Note that the communication between the *BPs* is ciphered using symmetric encryption. The symmetric keys are securely exchanged using asymmetric encryption.

**Transaction verification:** When a *BP* receives a transaction, it uses the smart contracts to verify its correctness. In the case of a node joining, the *BP* reruns the Assignment Algorithm to confirm that the node was assigned to the right set. It also checks if the node ID and the hash of the secret code have not already been used for another network node. On the other hand, if a node leaves the network or is evicted, the *BP* checks if the leaving node is actually a network member. It also verifies if there is a match between the node, subgroup and group *IDs* as well as the cryptographic hash of the node's secret code. If the *BP* judges that the transaction is correct, it adds it to its memory pool.

**Transaction validation:** After a certain period of time (*cp*) or when the size of the memory pool reaches a certain threshold (*ct*), the *BPs* run a consensus algorithm (see next section). The aim is to achieve a consensus between them on whether the content of the memory pool can be included to the blockchain. If all the *BPs* agree that a block of transactions can be added to the blockchain and when it is correctly stored, these transactions become valid. The layer 1 is therefore informed so that it can distribute the generated keys on the appropriate nodes.

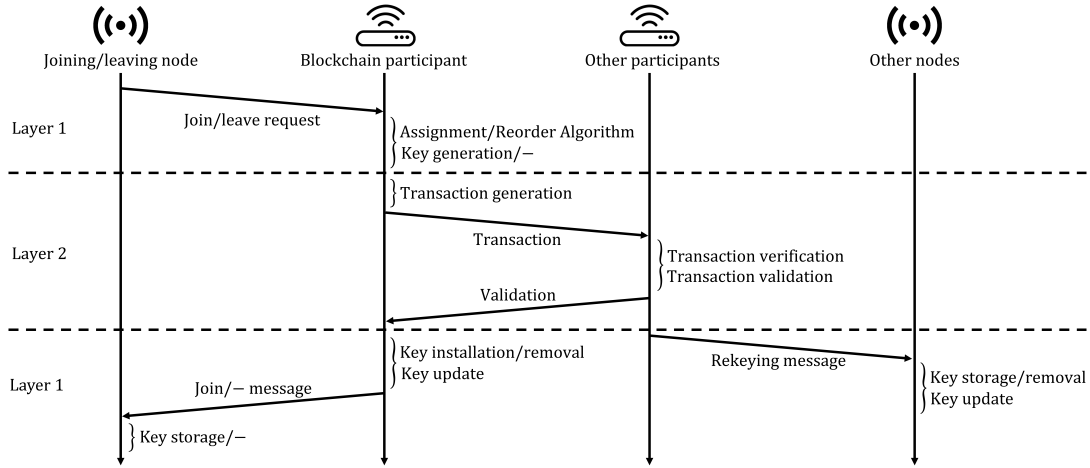


Figure 6.4 – Decentralized rekeying upon a network change using a blockchain.

### 6.3.2.2 Consensus Algorithm

We introduce a lightweight consensus algorithm for transaction validation. This is a proof of stake version that takes into account the capability of the blockchain participants, i.e. the *BP*'s capability determines its chance to validate the next block. In order not to favor the more powerful *BPs*, more parameters are included into the selection process: confidence and age. The level of confidence is initialized to a certain value, increased over time and decreased when an incorrect transaction from a *BP* is detected. The age of a *BP* represents the time elapsed between the last time it validated a block and a given time. This gives chance to the weak *BPs* to participate in the validation process when their age is high enough.

---

#### Algorithm 5: Consensus Algorithm

---

- 1 Generate a random number and broadcast it;
  - 2 Wait for the random numbers of the other *BPs*;
  - 3 Combine the received numbers into one value;
  - 4 Use this shared value to elect a validator;
  - 5 **if** the *BP* is the validator **then**
    - 6 Forge the new block;
    - 7 Sign the block;
    - 8 Broadcast the block;
  - 9 **else**
    - 10 Wait for the block from the validator;
    - 11 Check the block and its transactions;
  - 12 **end**
  - 13 Store the block in the blockchain;
  - 14 Remove validated transactions from memory pool;
  - 15 Update the validator parameters;
-

Our Consensus Algorithm (Algorithm 5) can be executed either periodically (the period is  $cp$ ) or when the number of transaction in the memory pool reaches a certain threshold ( $ct$ ). For each block, a validator (the  $BP$  that forges the block) is randomly elected by including a weighting according to the capabilities of  $BPs$  and the other above-mentioned parameters (the higher the capability, the confidence and the age of a  $BP$ , the higher its chance to be elected).

To achieve this, all  $BPs$  generate random numbers and exchange them with each other. Each of them combines all these random values by adding them, for example, or by applying another mathematical function. The resulting value, being common to all, can be used by  $BPs$  to perform a weighted random draw and thereby elect the same validator. This validator groups the transactions contained in its memory pool to forge the new block, signs it then broadcasts it. The other unelected  $BPs$  wait for the new block and check its content once received. All the  $BPs$ , including the validator, store the block in their copy of the blockchain and remove the validated transactions from their memory pool. Finally, they update the validator parameters. They reset its age and modify its level of confidence depending on whether an error is detected or not.

### 6.3.2.3 Blockchain interest

In addition to securely distribute the Key Management and ensure consistency between the different  $BPs$ , the use of blockchain offers functionalities that a centralized solution cannot provide. We present in the following the most important blockchain features, namely system availability, node mobility and node sleeping.

**System availability:** When a  $BP$  fails or when it is a target of malicious attacks (such as DoS attacks), the nodes attached to it become orphans. Each of them sends a rejoin request to another  $BP$ . When a  $BP$  receives a rejoin request, it agrees with the sender on new  $KEKs$  so they can securely communicate. After that, the node sends the hash of its secret code to be able to get authenticated. The  $BP$  consults the blockchain and checks if the hash received corresponds to that of the node  $ID$ . As the node in question is the only one able to generate the hash of its secret code, the  $BP$  concludes that it is really a network node. If it is the case, the node is then attached to this gateway without having to add new transactions to the blockchain (Figure 6.5). This makes the rejoin operation more efficient. More importantly, the failure of a  $BP$  does not prevent the system from working.

**Node mobility:** As when a *BP* fails, a node can use its secret code to get authenticated with another *BP* if its actual *BP* is no longer in range. The node sends a rejoin request to a *BP* which is within reach. When the *BP* receives the rejoin request, it sends to the node new *KEKs* to secure their communication. The node sends then the hash of its secret code to get authenticated. The *BP* consults the blockchain and checks if the hash received corresponds to that of the node *ID*. If it is the case, the node is attached to this gateway without adding new transactions to the blockchain (Figure 6.5). This makes the mobility operation more efficient. Our protocol is therefore well suitable for mobile networks.

**Node sleeping:** To save energy, a node can sleep if it does not have a work in progress. During sleeping, the node turns off its radio and will not receive the rekeying messages. Note that these messages contain the refresh keys that allow the network nodes to update their keys. Thus, the sleeping node will not have the opportunity to update its keys. However, when it wakes up, it will need the new keys to be able to securely communicate with the other network nodes. It will then send to its *BP* a rekey request containing the last refresh key it received. Since all the refresh keys are stored in the blockchain, the *BP* can retrieve and send to the node the refresh keys it missed. The keys will be updated without having to add new transactions to the blockchain.

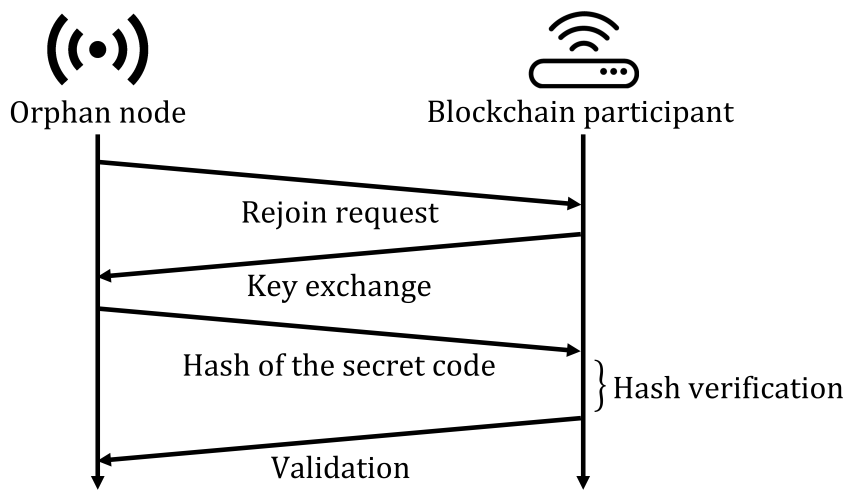


Figure 6.5 – Rejoin exchange.

## 6.4 Security analysis

In this section, we analyze the security of our solution. We start by showing that it guarantees the independence of services. Although resilience was detailed in the chapter on device-to-device communication (See section 4.4), we analyze it again here for two reasons. First, before the current chapter, the network was not divided into groups according to services. Moreover, our solution was centralized. we therefore assumed that the Key Manager itself is secure and that only the network nodes can be compromised. Thus, in this section, we evaluate the resilience of our solution against node capture (considering several groups) and against *BP* capture (considering that a *BP* can be compromised).

### 6.4.1 Independence of services

Using our solution, the security parameters of services are independent of each other. This is mainly due to the fact that the network is divided into groups and that nodes are distributed on them according to services in which they participate.

**Proposition 6.1:** Our solution ensures that the compromise of a service has no effect on the others.

**Proof:** All the keys, except the service keys, are specific to a single group and are independent from one group to another. The nodes belonging to the same group necessarily participate in the same services. Therefore, the compromise of one of them does not jeopardize a service in which it does not participate. Regarding the service keys, let us consider two groups  $G$  and  $I$  associated to the combinations of services  $A_{(G)}$  and  $A_{(I)}$ , respectively. When a member of  $G$  gets compromised, only the keys related to the services of  $A_{(G)}$  are exposed. If  $G$  and  $I$  share some services (i.e.  $A_{(G)} \cap A_{(I)} \neq \emptyset$ ), the keys related to the services to which the members of  $I$  participate, but not those of  $G$  ( $A_{(G)} \setminus A_{(I)}$ ), remain secret. This is because the compromised node does not know them. Furthermore, if the groups do not share services ( $A_{(G)} \cap A_{(I)} = \emptyset$ ), none of the keys related to the services of  $A_{(I)}$  get compromised. In both cases, only the services in which the captured node participates are exposed and the others remain secret.

### 6.4.2 Resilience against node capture

Although the heterogeneous algorithms that we presented in the previous chapter can be applied in this new solution, a homogeneous subgrouping allow us to evaluate resilience against node capture without a significant lack of generality. The  $n$  network nodes are then distributed into  $\sqrt{n}$  groups, each in turn divided into  $\sqrt[4]{n}$  subgroups of  $\sqrt[4]{n}$  nodes.

#### 6.4.2.1 Theoretical analysis

A node shares a pairwise key with each member of its subgroup, a pairwise subgroup key with every subgroup of its group and a pairwise service key with each group (whose members participate in the same service as the node).

**Lemma 6.1:** The number of links a node can decrypt is:

$$D = n - 1 + (\sqrt{n} - 1)(n - \sqrt{n}) + (\sqrt[4]{n} - 1)(\sqrt{n} - \sqrt[4]{n}) \quad (6.6)$$

**Proof:** A node can decrypt the communication linking it to the other  $n - 1$  network nodes in the worst case (if it participates in all the services). It may also decipher the messages exchanged between the  $\sqrt{n} - 1$  members of its group with the other  $n - \sqrt{n}$  network members. Finally, the node is able to decrypt the communication between the  $\sqrt[4]{n} - 1$  members of its subgroup with the  $\sqrt{n} - \sqrt[4]{n}$  other members of the group.

**Proposition 6.2:** The percentage of compromised links due to a node capture is:

$$P = 2 \frac{n - 1 + (\sqrt{n} - 1)(n - \sqrt{n}) + (\sqrt[4]{n} - 1)(\sqrt{n} - \sqrt[4]{n})}{n(n - 1)} \rightarrow 0, \text{ as } n \rightarrow \infty \quad (6.7)$$

**Proof:** From Lemma 6.1 and the fact that the total number of links is equal to  $C_n^2 = \frac{n(n-1)}{2}$ , we deduce this rate.

**Proposition 6.3:** The capture of the whole network requires the compromise of all the network nodes.

**Proof:** From Proposition 4.2, the capture of a single group requires the compromise of all its members. Moreover, deciphering all the inter-group communications requires the knowledge of all the pairwise service keys. This is only possible if at least a member of each group is captured.

### 6.4.2.2 Comparison

We compare the level of resilience offered by this new solution with that provided by the version presented in chapter 4. Figure 6.6 show that the percentage of comprised links due to a node capture is approximately the same for both solutions. Therefore, the division of the network into groups has no significant effect on the resilience of our protocol due to a node capture.

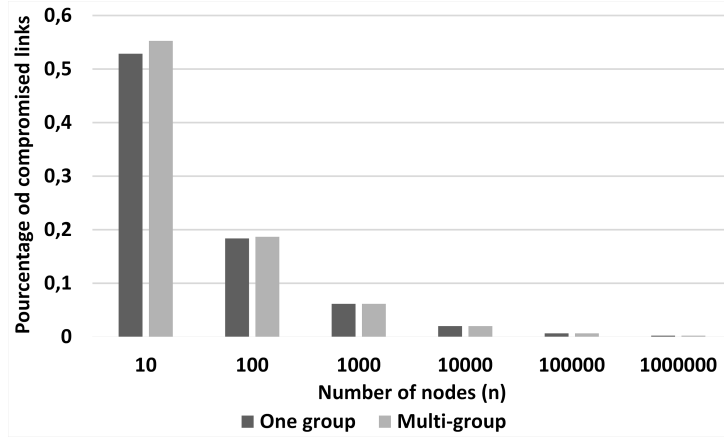


Figure 6.6 – Resilience against node capture.

### 6.4.3 Resilience against BP capture

In the following, we denote by  $r$  the number of  $BPs$  in the network. We then assume that the nodes are uniformly distributed on them (i.e.  $\frac{n}{r}$  nodes are attached to each  $BP$ ). This allows us to evaluate the resilience of our solution against  $BP$  capture without a significant lack of generality.

#### 6.4.3.1 Theoretical analysis

We study the rate of compromised links after  $BP$  capture.

**Proposition 6.4:** The percentage of links that a compromised  $BP$  can decipher is equal to:

$$P = \frac{D}{T} = \frac{2nr - n - r}{(n-1)r^2} \rightarrow \frac{2r-1}{r^2}, \text{ as } n \rightarrow \infty \quad (6.8)$$

**Proof:** A *BP* is responsible for the generation of the keys associated to the nodes attached to it. Therefore, if it gets compromised, it will be able to decipher the  $\frac{n}{2r}(\frac{n}{r} - 1)$  links between them. It will also be able to decipher the communications between its  $\frac{n}{r}$  nodes and the  $n - \frac{n}{r}$  other members of the network. It can then decrypt a total number of links equal to  $D = \frac{n}{2r}(\frac{n}{r} - 1) + \frac{n}{r}(n - \frac{n}{r})$ .

**Proposition 6.5:** The capture of the whole network requires the compromise of all the *BPs*.

**Proof:** From Proposition 6.3, deciphering all the communications requires the knowledge of all the pairwise keys. This is possible only if all the *BPs* are captured.

### 6.4.3.2 Comparison

We previously assumed that the Key Manager itself is secure and that only the network nodes can be compromised. In this chapter, we propose a decentralization based on the blockchain as in practice the central entity can be captured. Thanks to the blockchain features, the Key Management is securely decentralized so that the compromise of a *BP* has no effect on the others. Thus, compared to the solutions based on a centralized entity, which once captured the whole network is compromised, only a part is captured using our decentralized protocol (Figure 6.7). We showed that the rate of compromised links is inversely proportional to the number of *BPs*. In other words, the more we increase the number of *BPs*, the more resilient is our solution. In the next chapter, we analyze the effect of this parameter on the network performance to help the reader to choose the best compromise between resilience and performance.

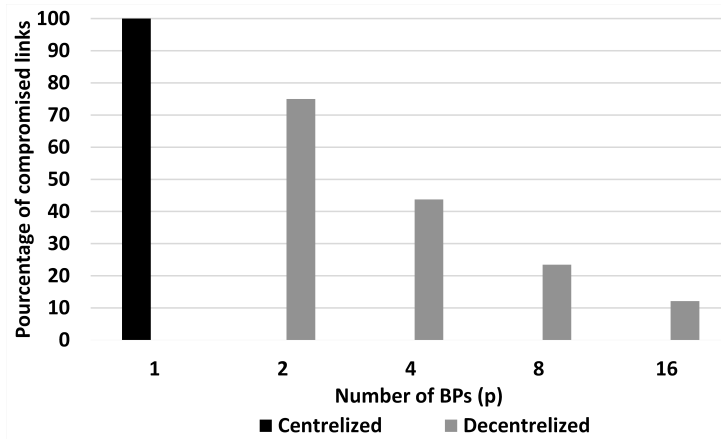


Figure 6.7 – Resilience against *BP* capture.



## 6.5 Performance evaluation

In this section, we evaluate the performance of our solution. After showing that our solution ensures the independence of services without significant loss of resilience, we prove that it remains efficient and highly scalable even with multiple groups. To achieve this, we focus in the layer 1 only. The blockchain management overheads are evaluated experimentally in the next chapter.

### 6.5.1 Overheads on the Key Manager

We start by analyzing the overheads of our solution on the Key Manager side.

**Property 6.1:** The communication overhead of an operation related to a subgroup  $S$  is proportional to the sum  $p + m^S + \sqrt{n}$  on the Key Manager.

**Proof:** Regardless of the rekeying operation performed (such as a node joining  $S$  or a node leaving  $S$ ), the Key Manager sends a unicast message to each of the  $m^S$  members of  $S$  and broadcasts a message for each of the other  $p - 1$  group subgroups as well as the other  $\sqrt{n} - 1$  groups, in the worst case. The Key Manager then sends a total number of messages proportional to the sum  $p + m^S + \sqrt{n}$ .

**Property 6.2:** The calculation overhead of an operation related to a subgroup  $S$  is proportional to the sum  $p + m^S + \sqrt{n}$  on the Key Manager.

**Proof:** Regardless of the rekeying operation performed (such as a node joining  $S$  or a node leaving  $S$ ), the Key Manager updates the keys which are or will be known by the node in question. The calculation overhead is therefore proportional to the storage cost on nodes, which will be proven to be of the order of  $p + m^S + \sqrt{n}$  in the next Section. Also, the subgroup management algorithms browse the list of sets in the worst case. They therefore have a complexity proportional to  $p$ .

**Property 6.3:** The storage overhead on the Key Manager is  $O(n)$ .

**Proof:** From Property 4.3 and 5.3, the storage cost on the Key Manager is  $O(n)$  when the network is not divided into groups. Two new types of keys are introduced in this chapter: service and pairwise service keys. The number of service keys is proportional to  $\sqrt{n}$ . The number of pairwise service keys is of the order on  $n$  as it corresponds to  $C_2^{[\sqrt{n}]}$ .

**Discussion:** The communication and calculation costs of an operation related to the subgroup  $S$  are proportional to the sum  $p + m^S + \sqrt{n}$  on the Key Manager. The storage, on the other hand, is of the order of  $n$ . Considering the significant improvement (presented in the next section) that our solution provides on the node side, the costs on the Key Manager are reasonable. Also, as the Key Manager has usually plentiful of resources, we aimed to make the costs more affordable on the nodes, especially the constrained ones.

### 6.5.2 Overheads on nodes

Now, we analyze the overheads of our solution on the node side.

**Property 6.4:** The communication overhead on the nodes is  $O(1)$ .

**Proof:** Regardless of the rekeying operation performed (such as a node joining or a node leaving), a node receives a constant number of messages. The communication cost on nodes is therefore independent of the network size.

**Property 6.5:** The calculation overhead of an operation related to a subgroup  $S$  is proportional to the sum  $p + m^S + \sqrt{n}$  on the nodes.

**Proof:** Regardless of the rekeying operation performed (such as a node joining  $S$  or a node leaving  $S$ ), a node updates all the keys it knows, in the worst case. The calculation cost on the nodes is therefore proportional to the storage, which will be proven to be proportional to the sum  $p + m^S + \sqrt{n}$  in the next proof.

**Property 6.6:** The storage overhead on a member of a subgroup  $S$  is proportional to the sum  $p + m^S + \sqrt{n}$ .

**Proof:** Using our protocol, a node of a subgroup  $S$  stores one secret key,  $m^S - 1$  pairwise node keys, one subgroup key,  $p - 1$  pairwise subgroup keys, one group key,  $k$  service keys and  $\sqrt{n}$  pairwise service keys. If we assume that the number of services in which the node participates ( $k$ ) is negligible compared to the other parameters, the storage overhead on the node will be proportional to the sum  $p + m^S + \sqrt{n}$ .

**Discussion:** The communication cost of our solution on the members of a subgroup  $S$  is  $O(1)$ , while the calculation and storage overheads are proportional to the sum  $p + m^S + \sqrt{n}$ . If a homogeneous subgrouping is used (i.e.  $\forall S, p = m^S = \sqrt{n}$ ), the calculation and storage costs of our solution will be  $O(\sqrt{n})$  on nodes. If a heterogeneous subgrouping is used, we can take into account an estimate of the value of  $\sqrt{n}$  in the calculation of the CEF. The heterogeneous Assignment and Reorder algorithms will then choose the subgroup sizes so that the constrained devices can handle the overheads.

---

## 6.6 Conclusion

The purpose of this chapter was to propose a novel decentralized blockchain-based Key Management protocol for the IoT. This new solution considers the three communication modes of the IoT: device-to-device, group and multi-group communication. Thus, unlike the solution presented in the previous chapters, this new one guarantees the secure coexistence of several services in a single network. To achieve this, it manages several groups having independent security parameters. Furthermore, this new solution is decentralized based on the blockchain technology and smart contracts. We then showed that it solves the single point of failure problem, since the system continues to operate when an entity fails. We also proved that resilience is improved, as the compromise of an entity does not jeopardize the whole network.

In the next chapter, we will propose an implementation of our solution considering the Contiki operating system and resource constrained IoT platforms. The aim is to experimentally complete the theoretical analyses presented in the current chapter and the previous ones.



# *Experimentation*

---

In this chapter, we propose an implementation of our solution on real IoT platforms with limited resources. The aim is to experimentally evaluate the performance of our solution and to complete the theoretical analyses of the previous chapters. We then start by presenting the software environment used as well as the hardware material in which our protocol was developed. In both cases, we consider the two components of our solution: the Key Manager and the nodes. We mainly focus on Contiki, which is an operating system designed for networked embedded devices. We also introduce the IoT motes on which we performed the experiments. Finally, we present the results of the tests carried out and compare them to those obtained by related works. The evaluated parameters include storage, computing and energy overheads of our protocol.

## 7.1 Introduction

In this thesis, we propose a novel decentralized blockchain-based Key Management protocol for secure communication between the heterogeneous and dynamic IoT devices. We showed, through theoretical studies, that our solution is well suitable for the IoT and meets the requirements of its devices. It also secures the three modes of IoT communication: device-to-device, group and multi-group communication. The purpose of this last chapter is to experimentally evaluate the performance of our protocol in order to complete the theoretical analyses presented in the previous chapters.

In the first part of this chapter, we present the software environment we used to implement our solution on the Key Manager and the node side. This includes the development environment, the security materials (the ciphers as well as the key derivation and hash functions) and the blockchain engine. After that, we introduce Contiki, which is a an operating system designed for networked embedded devices. Contiki provides a lightweight network stack for low-power wireless networks.

In the second part of the chapter, we present the hardware material on which we developed our protocol. Once again, we consider the two components of our solution: the Key Manager and the nodes. We mainly focus on the IoT platforms on which we performed the tests. We used various motes and most of them have limited resources. Thus, we were able to analyze the performance of our solution in heterogeneous networks containing constrained devices.

In the last part of the chapter, we introduce the results of the tests carried out and compare them to those obtained by related works. For the Key Manager, we evaluate the response time and compare it to that obtained using an other blockchain engine, namely Tendermint. For the nodes, we evaluate storage, computing and energy overheads of our protocol. The execution time and the energy consumption were obtained using the Powertrace tool.

The remainder of this chapter is organized as follows. Section 7.2 presents the software environment used to implement our solution. Section 7.3 details the Contiki operating system. Section 7.4 introduces the hardware material in which our protocol was developed. Section 7.5 presents our experimental platform. Section 7.6 introduces the results of the tests we obtained. Section 7.7 concludes the chapter.

---

## 7.2 Software environment

This first part of this chapter defines the software environment we used to develop our solution on the Key Manager and the node side. This includes the development environment, the security materials and the blockchain engine.

**Development environment:** We mainly worked on two IDEs: Pycharm and Code::Blocks. PyCharm is an integrated development environment for programming in Python, while Code::Blocks is for C. We utilized Python 3 to implement the Key Manager and we used C on the node side.

**Security material:** We mainly used two ciphers: ECC (Elliptic Curve cryptography) and AES (Advanced Encryption Standard). ECC is a public key encryption technique based on elliptic curve theory. It is based on smaller cryptographic keys, compared to other asymmetric protocols (e.g. RSA), which makes it faster and more efficient [86]. ECC allowed us to efficiently generate digital signatures and to perform key exchange between the *BPs*, i.e. on the Key Manager side. AES is the most widely used and most secure symmetric encryption algorithm available today [65]. We utilized it to secure the communication between nodes and the *BPs*. For the communication between the *BPs*, the keys have a lengths of 256 bits. Since some of the nodes we used only support AES-128, the key length of the keys used between them is of 128 bits. This provides an acceptable level of security according to the NIST [94]. We also used the CTR\_DRBG and HKDF for the key generation and derivation, respectively. Both are acceptable according to the NIST [15]. Note that HKDF is based on a hash-based message authentication code (HMAC). We then used SHA-2 as a hash function. Finally, all the above-mentioned protocols are supported by Python and C.

**Blockchain engine:** The blockchain layer was also implemented in Python 3. To evaluate the performance of our Consensus Algorithm, we compared it to an other blockchain application platform called Tendermint [64]. We mainly choose this platform for two reasons. First, it is a powerful blockchain engine based on the PBFT consensus algorithm (See section 2.4.4). Using Tendermint, hashing energy is not required to enter the next block. Therefore, compared to some of the most used consensus algorithms (e.g. PoW), PBFT reduces computation and thereby the energy consumption. Second, the application layer of Tendermint can be written in any programming language. It was then easy to use our solution (written in Python) in this environment, changing only the consensus algorithm (from our Consensus Algorithm to PBFT of Tendermint).

## 7.3 Operating system: Contiki

We used Ubuntu to implement the Key Manager and Contiki for the nodes. Contiki is a lightweight open-source operating system for resource-constrained devices [36]. Although being designed for networking applications, it can be used for other purposes. The Contiki applications are based on processes, its kernel is event-driven and its network stacks implement lightweight protocols. It also provides a simulator, called Cooja, which allows to simulate nodes that are Contiki compatible. According to the IoT developer survey conducted by Eclipse in 2019 [51], Contiki is one of the most used device operating systems.

### 7.3.1 Processes

A process is a function written in C, which usually contains an infinite loop and some blocking macro calls. During its execution, a process will run until it is blocked waiting for an event. There are multiple macros for the different blocking possibilities. Figure 7.1 presents the structure of a Contiki process [36]. It is a basic code that repeatedly waits for an event to occur.

```
#include "contiki.h"

/* The PROCESS() statement defines the process' name. */
PROCESS(my_example_process, "My example process");

/* The AUTOSTART_PROCESS() statement selects what process(es)
   to start when the module is loaded. */
AUTOSTART_PROCESSES(&my_example_process);

/* The PROCESS_THREAD() contains the code of the process. */
PROCESS_THREAD(my_example_process, ev, data)
{
    /* Do not put code before the PROCESS_BEGIN() statement -
       such code is executed every time the process is invoked. */
    PROCESS_BEGIN();
    /* Initialize stuff here. */
    while(1) {
        PROCESS_WAIT_EVENT();
        /* Do the rest of the stuff here. */
    }
    /* The PROCESS_END() statement must come at the end of the
       PROCESS_THREAD(). */
    PROCESS_END();
}
```

Figure 7.1 – Structure of a Contiki process.

### 7.3.2 Events

The Contiki kernel is event-driven, i.e. each part of the application is executed as a reaction to an event. There are mainly three kinds of events in the Contiki operating system:



**Timer events:** A timer can be used to generate an event after a given period of time.

The process is blocked, before the timer expires, and it continues its execution after.

**Internal events:** A process can address events to other processes. This is mainly used for inter-process communication.

**External events:** Peripheral devices (such as push-button, a radio chip or a sensor) may be connected to the microcontroller via Input/Output pins. These external devices can trigger interruptions to generate events in processes.

### 7.3.3 Network Stack

Contiki offers three lightweight network stacks: uIP for IPv4 networking, uIPv6 for IPv6 networking and Rime for low-power wireless networks. We used the Rime stack, which provides a hierarchical set of protocols. These protocols allow devices to send unicast or broadcast messages, anonymously or identified, in a single hop or a multi-hop communication. We mainly use the following modules (Figure 7.2):

**Abc:** This is the anonymous broadcast module. It broadcasts the packets via the radio driver and passes the received ones to the upper layer.

**Broadcast:** This is the identified broadcast module. It adds the sender address to the packets and passes them to the abc module.

**Unicast:** This module adds the destination address to the packets and passes them to the broadcast module. It also checks whether the destination addresses of the received packets match that of the node address.

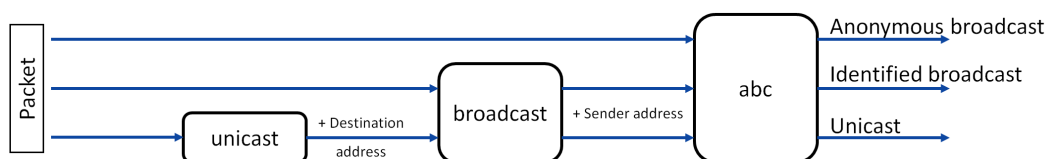


Figure 7.2 – Rime stack overview.

### 7.3.4 PowerTrace


Powertrace is a built-in tool used to analyze the energy consumption of a sensor node. It reports the resource utilisation of a node and prints the statistics to the console. This tool is accurate to 94% of the real energy consumption of a device [42].

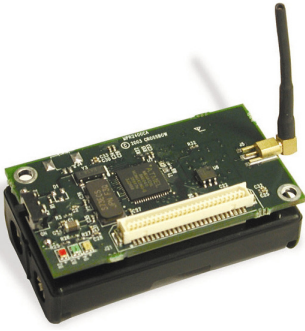

### 7.3.5 Cooja

Cooja (acronym of COnтики Operating system JAVa simulator) is the Contiki network simulator [4]. It allows to simulate large and small networks of Contiki motes. Offering a variety of nodes, it is also possible to simulate heterogeneous networks. As the number of physical devices we possess is limited, we used Cooja to get larger networks.

## 7.4 Material resource

For our experiments, we used a laptop and IoT motes as hardware resources. The laptop is an Intel Core i7 with 4GB RAM. The IoT motes are of 5 types (all based on Contiki OS): Exp5438 [91], MicaZ [90], Openmotes [98], TelosB [125] and Z1 [126]. Given the limited number of physical motes available to us, some are Cooja motes. Table 7.1 summarizes the specifications of each of the IoT motes used as well as their number.

Mote	Description
 <p>Exp5438</p>	<p><b>Microcontroller:</b></p> <ul style="list-style-type: none"> <li>• <b>Family:</b> MSP430F5438</li> <li>• <b>RAM:</b> 16 kB</li> <li>• <b>Flash:</b> 256 kB</li> <li>• <b>Frequency:</b> 18 MHz</li> </ul> <p><b>Radio transceiver:</b></p> <ul style="list-style-type: none"> <li>• <b>Frequency band:</b> 2.4 GHz</li> <li>• <b>Standard compliance:</b> IEEE 802.15.4 compliant</li> <li>• <b>Transfer Rate:</b> 250 kbps</li> </ul> <p><b>Applications:</b></p> <ul style="list-style-type: none"> <li>• Energy harvesting</li> <li>• Automatic metering infrastructure (AMI)</li> </ul> <p><b>Number:</b> 22 Cooja motes.</p>

 <p>MicaZ</p>	<p><b>Microcontroller:</b></p> <ul style="list-style-type: none"> <li>• <b>Family:</b> ATMEGA128L</li> <li>• <b>RAM:</b> 4 kB</li> <li>• <b>Flash:</b> 128 kB</li> <li>• <b>Frequency:</b> 16 MHz</li> </ul> <p><b>Radio transceiver:</b></p> <ul style="list-style-type: none"> <li>• <b>Frequency band:</b> 2.4 to 2.4835 GHz</li> <li>• <b>Standard compliance:</b> IEEE 802.15.4 compliant</li> <li>• <b>Transfer Rate:</b> 250 kbps</li> </ul> <p><b>Applications:</b></p> <ul style="list-style-type: none"> <li>• Indoor Building Monitoring and Security</li> <li>• Large Scale Sensor Networks</li> </ul> <p><b>Number:</b> 10 physical motes and 12 Cooja motes.</p>
 <p>OpenMote</p>	<p><b>Microcontroller:</b></p> <ul style="list-style-type: none"> <li>• <b>Family:</b> ARM Cortex-M3</li> <li>• <b>RAM:</b> 32 kB</li> <li>• <b>Flash:</b> 512 kB</li> <li>• <b>Frequency:</b> 32 MHz</li> </ul> <p><b>Radio transceiver:</b></p> <ul style="list-style-type: none"> <li>• <b>Frequency band:</b> 2.4 GHz</li> <li>• <b>Standard compliance:</b> IEEE 802.15.4 compliant</li> <li>• <b>Transfer Rate:</b> 250 kbps</li> </ul> <p><b>Applications:</b></p> <ul style="list-style-type: none"> <li>• Home and Building Automation</li> <li>• Intelligent Lighting Systems</li> </ul> <p><b>Number:</b> 12 physical motes.</p>


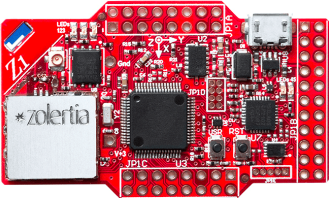
 <p>TelosB</p>	<p><b>Microcontroller:</b></p> <ul style="list-style-type: none"> <li>• <b>Family:</b> MSP430</li> <li>• <b>RAM:</b> 10 kB</li> <li>• <b>Flash:</b> 48 kB</li> <li>• <b>Frequency:</b> 8 MHz</li> </ul> <p><b>Radio transceiver:</b></p> <ul style="list-style-type: none"> <li>• <b>Frequency band:</b> 2.4 to 2.4835 GHz</li> <li>• <b>Standard compliance:</b> IEEE 802.15.4 compliant</li> <li>• <b>Transfer Rate:</b> 250 kbps</li> </ul> <p><b>Applications:</b></p> <ul style="list-style-type: none"> <li>• Platform for Low Power Research Development</li> <li>• Wireless Sensor Network Experimentation</li> </ul> <p><b>Number:</b> 6 physical motes and 16 Cooja motes.</p>
 <p>Z1</p>	<p><b>Microcontroller:</b></p> <ul style="list-style-type: none"> <li>• <b>Family:</b> MSP430F2617</li> <li>• <b>RAM:</b> 8 kB</li> <li>• <b>Flash:</b> 92 kB</li> <li>• <b>Frequency:</b> 16 MHz</li> </ul> <p><b>Radio transceiver:</b></p> <ul style="list-style-type: none"> <li>• <b>Frequency band:</b> 2.4 GHz</li> <li>• <b>Standard compliance:</b> IEEE 802.15.4 compliant</li> <li>• <b>Transfer Rate:</b> 250 kbps</li> </ul> <p><b>Applications:</b></p> <ul style="list-style-type: none"> <li>• Personal healthcare monitoring</li> <li>• Environmental monitoring</li> </ul> <p><b>Number:</b> 22 Cooja motes.</p>

Table 7.1 – IoT mote specifications.

## 7.5 Experimental platform

Our solution consists of three components: the Key Manager (Laptop), the nodes (IoT motes) and intermediate motes to connect the two. The nodes communicate with intermediate motes using their radios and the intermediate motes communicates with the laptop using serial ports (Figure 7.3).

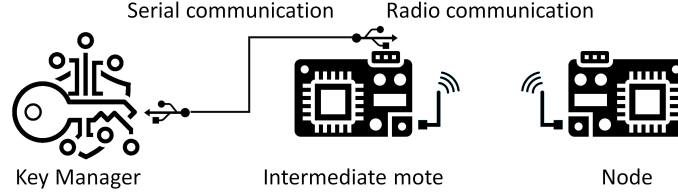


Figure 7.3 – Network components.

### 7.5.1 Key Manager

The Key Manager is represented by a set of *BPs*, all implemented on the laptop using Python. Their number varies between 2 and 16 according to the test performed. Each *BP* listens to a different serial port, waiting for node join requests (from physical motes, Cooja motes or the request simulator). When a *BP* receives a request, it processes it according to the steps described in the previous chapters. Note that we implemented a basic version using no hash function, for the pairwise keys, and considering only one group. Once the processing is finished, the *BP* sends the keys via its serial port (Figure 7.4).

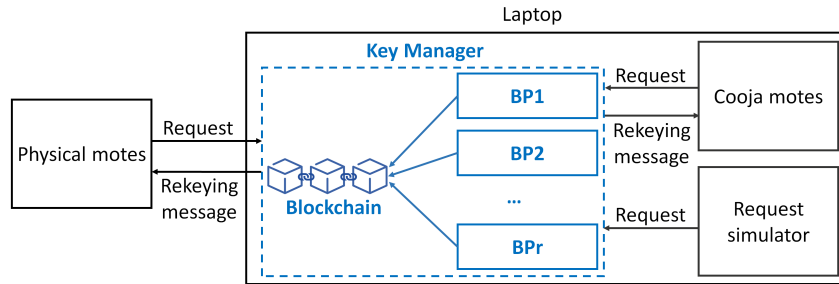


Figure 7.4 – Key Manager.

### 7.5.2 Nodes

As stated in Table 7.1, we used 100 heterogeneous nodes, all within the reach of intermediate nodes (routing is therefore not necessary). Although some are real and other are Cooja motes (Figure 7.4), they belong to the same network. After all the nodes joined the network, we obtained the subgroup distribution shown in Figure 7.5. The subgroups 4 and 5 result from the split of the subgroups 2 and 1 (See section 5.3.2.3), respectively.

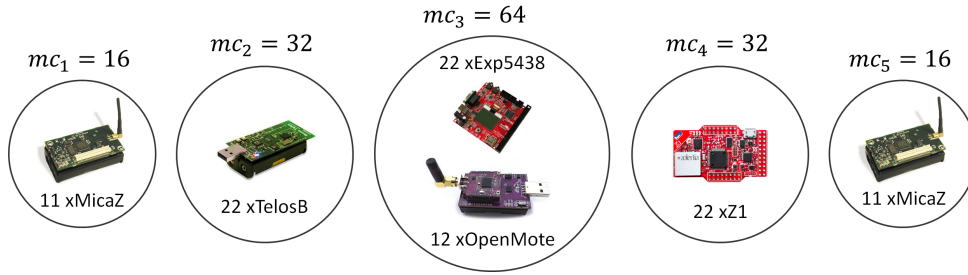


Figure 7.5 – Network partitioning.

Each node executes a process that started by sending a join request to a *BP*. After a node joins the network, its process constantly awaits for rekeying messages (external events). When it receives a message, the process updates the security materials according to the steps described in the previous chapters. Although Cooja can simulate large networks, after a certain threshold its performance deteriorate considerably. Therefore, some requests are simulated to perform the tests in which we need more nodes (between 100 and 1000). These requests are processed by *BPs* and the resulting messages are sent to the physical and the Cooja nodes. This allows us to evaluate the performance of our solution (the processing of requests by the *BPs*) in larger networks.

### 7.5.3 Intermediate notes

The intermediate notes are notes that make the link between the manager and the nodes. We used 5 intermediate notes, one for each type of node (Exp5438, MicaZ, OpenMote, TelosB and MicaZ). Some of them are physical, while others are Cooja notes. Physical intermediate notes are directly connected to the usb ports of the laptop. They communicate with the physical nodes using their radios and with the laptop using serial ports. Cooja intermediate notes are used to link the Cooja nodes to the Key manager. Our experimental platform is parented in Figure 7.6.

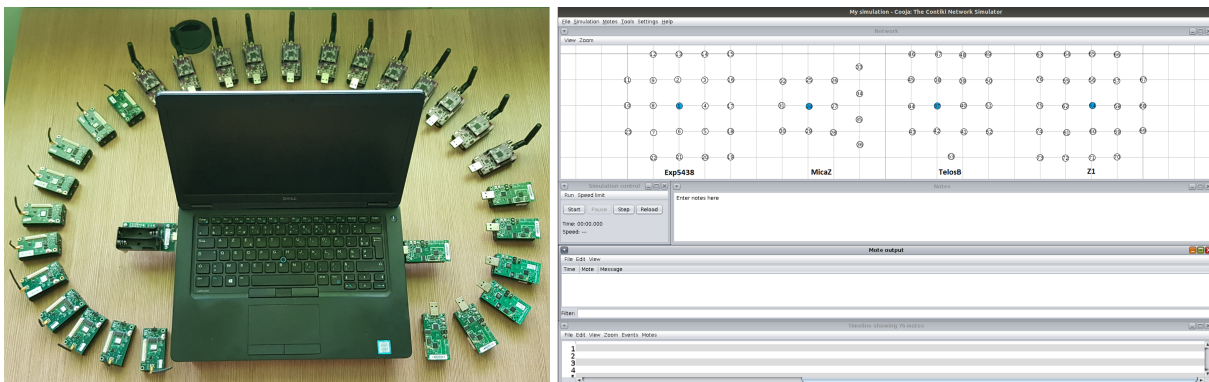


Figure 7.6 – Experimental platform.

## 7.6 Experimental results

Now, we present the results of the tests carried out. For the Key Manager, we consider the response time. This is the time separating the reception of a request by a *BP* from the sending of a response. For the nodes, we consider the storage cost as well as the computing time and the energy consumption of rekeying operation.

### 7.6.1 Response time of *BPs*

We send to the *BPs* a certain number of requests at the same time and calculate the response time for the first and the last request. We analyze the effect of three parameters on the response time: the number of *BPs* ( $r$ ), the number of simultaneous requests ( $nst$ ) and the consensus period ( $cp$ ). Each time we set two parameters to default values and we vary the third. The default values of the three parameters are 4, 100 and 10 ms, respectively. The size of the memory pool ( $ct$ ) is set to the number of *BPs* ( $r$ ). After that, we compare the results obtained using our Consensus Algorithm (See section 6.3.2.2) to that of Tendermint.

**Number of *BPs*:** The results (Figure 7.7) show that the more the number of *BPs* increases, the more the processing time of one request rises. This can be explained by the fact that there is more communication between *BPs*. However, if more than one request are received at the same time, we notice a decrease in the response time (of the last request) before it starts going up again. This is because several *BPs* can process different requests at the same time. However, after a certain threshold (8 *BPs* with our means), the time lost due to communications covers the time saved thanks to parallelism.

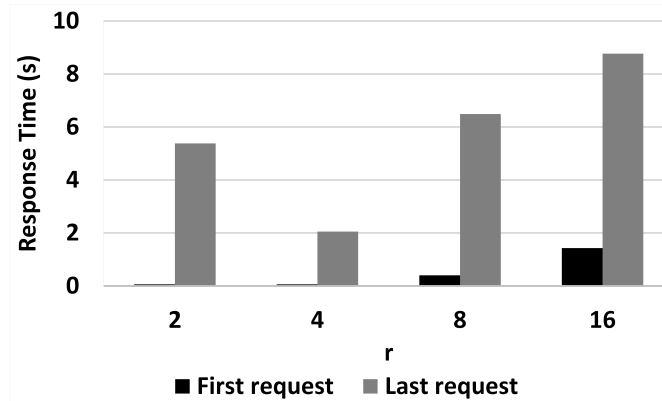


Figure 7.7 – Effect of  $r$  on response time.

**Number of simultaneous requests:** The results (Figure 7.8) show that the more the system receives simultaneous requests, the more the response time increases, especially for the last request. Note that with our means, more than 1000 requests are processed per minute.

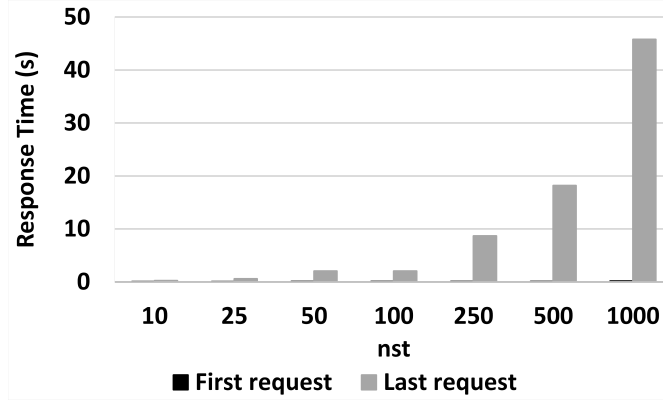


Figure 7.8 – Effect of *nst* on response time.

**Consensus period:** The best results (Figure 7.9) are obtained when the period to forge new blocks is neither too short nor too long (10 ms with our means). If it is too short, there will be a lot of unnecessary message exchanges, while the memory pool is empty. Conversely, when this period is too long, the processing time of a request increases.

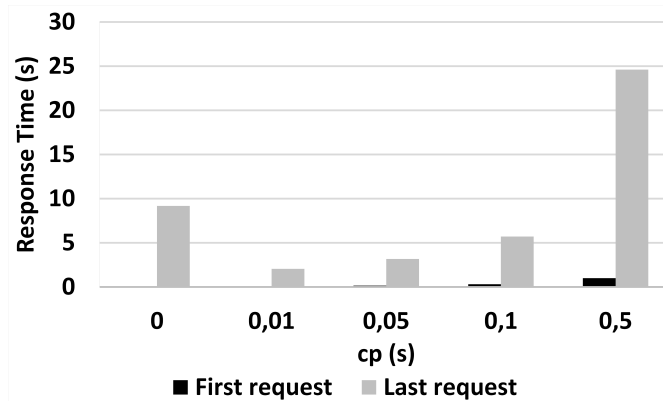


Figure 7.9 – Effect of *cp* on response time.

**Consensus Algorithm:** We finally compare our consensus algorithm to Tendermint. Note that we have not modified any of the default settings of Tendermint. The other parameters that are not related to the consensus algorithms (such as the key length, the assignment algorithm and the material used) are the same in both cases (using our consensus algorithm and PBFT of Tendermint). The obtained results are plotted in Figure 7.10. They show that regardless of the number of *BPs*, the process of one request is always faster using our solution compared to Tendermint.



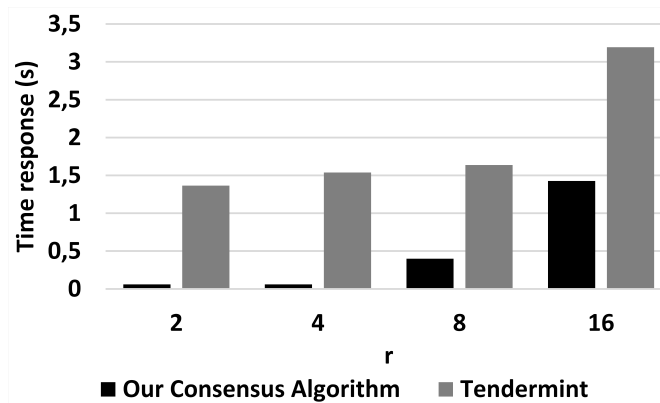


Figure 7.10 – Comparison of our consensus algorithm with Tendermint.

### 7.6.2 Storage overhead on nodes

We assume that the Key Manager has plentiful of resources and focus on the storage costs on nodes. The program and the values of the initialized variables are stored in the flash memory, while the data (including the cryptographic keys) are saved in RAM. The number of keys stored in nodes varies depending on whether a heterogeneous (proportional to the node capacities) or a homogeneous subgrouping is used ( $\sqrt{n}$ ). For comparison, we also consider the case where no subgrouping is used (i.e. nodes hold  $n$  keys). The memory occupation in kilobytes and in percentage is presented in Table 7.2.

Node	Flash	RAM		
		Het	Hom	None
Exp5438	25000 kB (9.5%)	9234 kB (56.36%)	8962 kB (54.70%)	10226 kB (62.41%)
MicaZ	24500 kB (18.69%)	3980 kB (97.17%)	4076 kB (99.51%)	5340 kB (130.37%)
OpenMote	28047 kB (5.35%)	7498 kB (22.85%)	7217 kB (22.02%)	8481 kB (25.88%)
TelosB	30872 kB (62.81%)	5228 kB (51.05%)	5148 kB (50.27%)	6412 kB (62.62%)
Z1	32290 kB (34.28%)	3490 kB (42.6%)	3410 kB (43.63%)	6474 kB (57.06%)

Table 7.2 – Storage overhead on nodes (Keys stored in the RAM).

The size of the flash memory is larger than that of the RAM for most of the motes used, especially the MicaZ. As a result, a considerable part of the RAM is used, while there is more space in the flash memory. We then saved the keys in files so that they are stored in the flash memory. To achieve this, we used the Coffee File System of Contiki [37]. Coffee is used with resource-constrained devices equipped with flash memories. The memory occupation, in this case, is presented in Table 7.3. The results show that memory usage becomes more balanced between the two types of memories and relieves the RAM.

Node	Flash			RAM
	Het	Hom	None	
Exp5438	25592 kB (9.76%)	25320 kB (9.66%)	26584 kB (10.14%)	8642 kB (52.75%)
MicaZ	24900 kB (19%)	24820 kB (18.94%)	26084 kB (19.9%)	3756 kB (91.70%)
OpenMote	28639 kB (5.46%)	28367 kB (5.41%)	29631 kB (5.65%)	6897 kB (21.05%)
TelosB	31096 kB (63.26%)	31192 kB (64.46%)	31096 kB (66.03%)	4828 kB (47.15%)
Z1	32690 kB (34.7%)	32610 kB (34.61%)	33874 kB (35.96%)	3090 kB (37.72%)

Table 7.3 – Storage overhead on nodes (Keys stored in the flash).

By analyzing the results, the occupation of the RAM memory of the MicaZ motes may seem important. However, it is important to note that just the basic communication program, which consists of periodically sending and receiving unicast messages (without the broadcast code or the AES encryption code), occupies 67% of the RAM. In other words, a large part of the memory is used by other process (mostly networking process). The part of the memory occupied by our Key Management protocols remains then reasonable, especially considering the very limited amount of RAM memory the MicaZ motes have.

We consider the work presented in [109] as an example for comparison. The network is composed of 15 TelosB motes only and yet 96.3% (against 63.26% using our solution) of their flash memory and 74.92% (against 47.15% using our solution) of their RAM are occupied. Therefore, our solution requires less space on both memory types, at least on TelosB motes.

### 7.6.3 Execution time on nodes

The evaluation of the computing overhead of our solution concerns two operations a node may perform: key installation and key update. The key installation corresponds to the operation by which a new node, which has just join the network, processes the Key Manager’s messages and stores the keys assigned to it in its memory. The key update is the operation by which a network node updates its keys upon a network change. For this operation, we consider both cases where a subgroup is split or not. The execution time of any of these operations corresponds to the time separating the reception of a message from a *BP* and the end of its processing. The execution times on nodes are presented in Table 7.4. The results were obtained using Powertrace. This tool is not supported on MicaZ. Therefore, we only present the execution time on the other motes.

Node	Key installation	Key update	Key update with split
Exp5438	46.02 ms	60.27 ms	137.02 ms
OpenMote	100.89 ms	102.47 ms	209.41 ms
TelosB	73.36 ms	86.73 ms	177.25 ms
Z1	58.01 ms	71.65 ms	163.27 ms

Table 7.4 – Execution time on nodes (Keys stored in the RAM).

The processing times are shorter when keys are stored in RAM because the access to flash memory, which is slower, is reduced. Despite this, we still present, in Table 7.5, the execution times when the keys are saved in flash memory. Indeed, although storing keys in flash memory slows down the protocol, we previously showed that this relieves the RAM when it is limited.

Node	Key installation	Key update	Key update with split
Exp5438	57.98 ms	69.18 ms	139.34 ms
OpenMote	101.86 ms	103.46 ms	211.46 ms
TelosB	85.41 ms	89.02 ms	177.28 ms
Z1	60.33 ms	76.29 ms	167.23 ms

Table 7.5 – Execution time on nodes (Keys stored in the flash).

#### 7.6.4 Energy consumption by nodes

For energy consumption, we consider the same operations as in the previous section (key installation and key update with or without subgroup split). The energy consumed by a node includes calculations and communications. We then used Powertrace to evaluate the energy consumption of our solution per second. The results are presented in Table 7.6.

Node	Key installation	Key update	Key update with split
Exp5438	2.65 mJ	3.08 mJ	7.5 mJ
OpenMote	4.38 mJ	4.5 mJ	10.01 mJ
TelosB	1.64 mJ	4.08 mJ	8.2 mJ
Z1	3.52 mJ	4.12 mJ	9.45 mJ

Table 7.6 – Energy consumption by nodes (Keys stored in the RAM).

For the same reasons as for the execution time, we present in Table 7.7 the results of the energy consumption by nodes when the keys are stored in the flash memory. We notice that the consumption is slightly higher.

Node	Key installation	Key update	Key update with split
Exp5438	3.2 mJ	3.95 mJ	8.41 mJ
OpenMote	4.38 mJ	4.52 mJ	10.03 mJ
TelosB	2.32 mJ	4.22 mJ	9.83 mJ
Z1	3.7 mJ	4.38 mJ	9.46 mJ

Table 7.7 – Energy consumption by nodes (Keys stored in the flash).

In Table 7.8, we present an estimate of the lifespan of 2xAA batteries using our solution. The lifespan is calculated, for each type of node, according to the average value of the quantity of energy it consumes. The results are obtained assuming that the nodes are constantly receiving rekeying messages.

Node	Battery life
Exp5438	101 days
OpenMote	69 days
TelosB	74 days
Z1	76 days

Table 7.8 – Battery life.

### Discussion:

Although some authors (such as those of [19, 87, 97]) evaluate the energy consumption of their Key Management protocols, they are based on theoretical models and cannot be compared to our experimental results. An example of an energy consumption model can be found in [57]. In the absence of similar works presenting the results of an implementation on the same IoT motes we used, we consider the results presented in [114]. The authors of the document use Powertrace to estimate the energy consumption of TelosB motes when sending ipv6 packets (while we used the lightweight Rime Stack). Since communication is the operation that consumes the most energy, these results can be compared to the energy consumption of our solution, even if they do not consider the overheads related to the Key Management.

---

## 7.7 Conclusion

The purpose of this chapter was to validate our theoretical analyses by implementing our protocol on real IoT platforms. The IoT motes used are based on Contiki, which is an operating system designed for networked embedded devices. We then started by presenting the software environment used to implement the Key Manager and the nodes. Next, we detailed the hardware material in which these two components of our solution were developed. Finally, we presented the results of the tests carried out. The evaluated parameters include storage, computing and energy overheads of our protocol.



## *Conclusion and future works*

---

The Internet of Things is changing much about the world we live in. It is reshaping the way we shop, drive, learn and even how we practise sport and have fun. Sensors, chips and antennas are increasingly embedded in everyday things. This gives them the ability to collect data from their surrounding and transmit them to other connected objects. The recorded data are then automatically processed and depending on the results these things can in turn act on their environment. This may be used, for example, to improve the production of a factory, give city residents real-time updates on where to park or monitor our personal health.

Thanks to these smart devices, the Internet of Things is creating a kind of bridge between the physical and the digital worlds. A bridge that will allow us to cross new horizons and discover a new world. However, many challenges are slowing down its development. There are mainly three reasons behind this. The first one is the increasing number of the connected devices. This results in a tremendous amount of data. It is therefore often very difficult and costly to manage them. The second reason is the very nature of the devices of the Internet of Things. They are indeed mobile, wireless and physically small. Therefore, they have little storage, low computing power and limited energy supply. Finally, the heterogeneous nature of these smart objects makes their collaboration too complicated.

One of the most important issues facing the Internet of Things is how to secure communication between its devices. Although cryptography has proven itself as effective in the traditional Internet, it is not efficient in the Internet of Things. For the reasons mentioned above, the cryptography mechanisms can hardly or not at all be implemented in the Internet of Things. They have been indeed designed for desktop/server environments and not for limited-resource devices. To tackle this problem, it is necessary to find lightweight alternative solutions that are more suitable for use in constrained environments. These new solutions must provide a good compromise between security, performance and resource requirements (hardware cost). This relatively new area of research is referred to as lightweight cryptography.

In this thesis, we proposed a novel lightweight Key Management protocol for the Internet of Things. It is important to note that, among all the security issues, the Key Management is one of the most important and the most challenging. The main role of such a system is to establish secure links between the communicators. To achieve this, it provides them with secret cryptographic keys that are used to secure communication. The Key Management is responsible of the generation, the storage, the distribution and the replacement of these keys. Being a branch of cryptography, the Key Management solutions for the Internet of Things must also be lightweight to respond to its needs.

Our solution secures the three communication modes: device-to-device, group and multi-group communication. We then showed that it fulfils the requirements of each of them, while most of the related works focus on one of them only. For device-to-device communication, our solution provides a good level of resilience. It also guarantees a total connectivity coverage and supports device mobility. Regarding group communication, our solution ensures the backward and forward secrecy and resists to collusion. It also guarantees the independence of services for a secure multi-group communication.

Furthermore, unlike most of the existing solutions proposed for the traditional Internet and static wireless networks, our solution considers the heterogeneous and dynamic nature of the Internet of Things. It balances the loads between devices according to their capabilities. We showed that this makes our solution efficient and highly scalable. Our solution is also flexible as it allows devices to securely join and leave the network at any time. The cryptographic materials will then be automatically updated and distributed to the network members.

Our solution is finally decentralized as computation and storage can be distributed over multiple entities. It is therefore fault-tolerant and does not require the trust of a single third party for the handling of keys. To reach this goal, we used the blockchain technology and smart contracts. We then showed that the system will continue to operate even if an entity fails and that the compromise of an entity will not jeopardize the security of the whole network.

To conclude, we proposed, in this work, a novel decentralized blockchain-based Key Management protocol for secure communication between the heterogeneous and dynamic IoT devices. Through theoretical studies, simulations and an experimentation platform, we were able to show that our solution is well suitable for the Internet of Things and responds to the requirements of its devices. The contributions of this thesis are very encouraging and open up multiple research perspectives.



---

First, lightweight cryptography still being an open issue, we intend to extend the heterogeneous strategy we propose for the Key Management to other cryptographic mechanisms. In encryption, for example, better results can be obtained if, in addition to the Key Management, the ciphers also take into account the capabilities of the devices. The constrained one will then use little resources for handling the keys and ciphering the data. Even public cryptography may get closer to lightweight cryptography if it takes advantage of the heterogeneous nature of the Internet of Things.

Moreover, we intend to design key agreement methods that are more efficient than the existing ones. This will improve our solution, especially the joining process. Indeed, a key establishment is necessary to secure the first communication with a device that wishes to join the network. Besides, the less it consumes resources, the more efficient our solution becomes.

Finally, many security mechanisms suffer from their dependence of a third party for their proper functioning. The blockchain technology and smart contract offer an alternative, which is encouraged by the results of our solution. Given their complementary features, the blockchain and the Internet of Things should supplement one another to face the multiple challenges that are slowing down their development. We then plan to dig further to discover the potential that the combination of the two technologies can offer.



---

# *Bibliography*

---

- [1] Priyanka Ahlawat and Mayank Dave. “A cost-effective attack matrix based key management scheme with dominance key set for wireless sensor network security”. In: *International Journal of Communication Systems* 31.12 (2018), e3713.
- [2] Priyanka Ahlawat and Mayank Dave. “An attack model based highly secure key management scheme for wireless sensor networks”. In: *Procedia Computer Science* 125 (2018), pp. 201–207.
- [3] Sofiane Aissani, Mawloud Omar, Abdelkamel Tari, and Ferial Bouakkaz. “ $\mu$ KMS: micro key management system for WSNs”. In: *IET Wireless Sensor Systems* 8.2 (2018), pp. 87–97.
- [4] *An Introduction to Cooja*. 2019. URL: <https://github.com/contiki-os/contiki/wiki/An-Introduction-to-Cooja>.
- [5] HS Annapurna and M Siddappa. “A Technique for Multi-tier Key Distribution for Securing Group Communication in WSN”. In: *Emerging Research in Computing, Information, Communication and Applications*. Springer, 2015, pp. 273–279.
- [6] Arafat MA Ansari. *Smart vehicle*. US Patent 9,711,050. July 2017.
- [7] Frederik Armknecht, Ghassan O Karame, Avikarsha Mandal, Franck Youssef, and Erik Zenner. “Ripple: Overview and outlook”. In: *International Conference on Trust and Trustworthy Computing*. Springer. 2015, pp. 163–180.
- [8] E. Baburaj et al. “Polynomial and multivariate mapping-based triple-key approach for secure key distribution in wireless sensor networks”. In: *Computers & Electrical Engineering* 59 (2017), pp. 274–290.
- [9] Kannan Balasubramanian. “Hash Functions and Their Applications”. In: *Algorithmic Strategies for Solving Complex Problems in Cryptography*. IGI Global, 2018, pp. 66–77.
- [10] Arati Baliga, I Subhod, Pandurang Kamat, and Siddhartha Chatterjee. “Performance evaluation of the quorum blockchain platform”. In: *arXiv preprint arXiv:1809.03421* (2018).

- 
- [11] N. Baracaldo, B. Palanisamy, and J. Joshi. “G-sir: an insider attack resilient geo-social access control framework”. In: *IEEE Transactions on Dependable and Secure Computing* (2017).
  - [12] Elaine Barker. *Recommendation for key management: Part 1: General*. National Institute of Standards and Technology, Technology Administration, 2020.
  - [13] Elaine Barker and William Barker. *Recommendation for Key Management, Part 2: Best Practices for Key Management Organization*. Tech. rep. National Institute of Standards and Technology, 2018.
  - [14] Elaine Barker and Allen Roginsky. *Recommendation for cryptographic key generation*. US Department of Commerce, National Institute of Standards and Technology, 2019.
  - [15] Elaine Barker and Allen Roginsky. *Transitioning the use of cryptographic algorithms and key lengths*. Tech. rep. National Institute of Standards and Technology, 2018.
  - [16] Elaine Barker, Miles Smid, Dennis Branstad, and Santosh Chokhani. “A framework for designing cryptographic key management systems”. In: *NIST Special Publication* 800.130 (2013), pp. 1–112.
  - [17] Imran Bashir. *Mastering blockchain*. Packt Publishing Ltd, 2017.
  - [18] Lawrence Bassham, Çağdaş Çalık, Kerry McKay, and Meltem Sönmez Turan. *Submission requirements and evaluation criteria for the lightweight cryptography standardization process*. Tech. rep. Technical report, US National Institute of Standards and Technology, 2018.
  - [19] Walid Bechkit, Yacine Challal, Abdelmadjid Bouabdallah, and Vahid Tarokh. “A highly scalable key pre-distribution scheme for wireless sensor networks”. In: *IEEE Transactions on Wireless Communications* 12.2 (2013), pp. 948–959.
  - [20] Simon Blake-Wilson, Nelson Bolyard, Vipul Gupta, Chris Hawk, and Bodo Moeller. *Elliptic curve cryptography (ECC) cipher suites for transport layer security (TLS)*. Tech. rep. RFC 4492, May, 2006.
  - [21] Rolf Blom. “An optimal class of symmetric key generation systems”. In: *Workshop on the Theory and Application of Cryptographic Techniques*. Springer. 1984, pp. 335–338.
  - [22] Richard Gendal Brown, James Carlyle, Ian Grigg, and Mike Hearn. “Corda: an introduction”. In: *R3 CEV, August* 1 (2016), p. 15.

- 
- [23] William J Buchanan, Shancang Li, and Rameez Asif. “Lightweight cryptography methods”. In: *Journal of Cyber Security Technology* 1.3-4 (2017), pp. 187–201.
  - [24] Christian Cachin et al. “Architecture of the hyperledger blockchain fabric”. In: *Workshop on distributed cryptocurrencies and consensus ledgers*. Vol. 310. 2016, p. 4.
  - [25] Miguel Castro, Barbara Liskov, et al. “Practical Byzantine fault tolerance”. In: *OSDI*. Vol. 99. 1999. 1999, pp. 173–186.
  - [26] Tafadzwa Tapuwa Chakavarika, Shashi Kant Gupta, and Brijesh Kumar Chaurasia. “Energy efficient key distribution and management scheme in wireless sensor networks”. In: *Wireless Personal Communications* 97.1 (2017), pp. 1059–1070.
  - [27] Haowen Chan, Adrian Perrig, and Dawn Song. “Random key predistribution schemes for sensor networks”. In: *Symposium on Security and Privacy, 2003. Proceedings*. IEEE. 2003, pp. 197–213.
  - [28] Che-Yu Chang, Hsu-Chun Yen, and Der-Jiunn Deng. “V2V QoS guaranteed channel access in IEEE 802.11 p VANETs”. In: *IEEE Transactions on Dependable and Secure Computing* 13.1 (2016), pp. 5–17.
  - [29] Kakali Chatterjee, Asok De, and Daya Gupta. “An improved ID-Based key management scheme in wireless sensor network”. In: *Int. Conf. in Swarm Intelligence*. Springer. 2012, pp. 351–359.
  - [30] Jaewoo Choi, Jihyun Bang, LeeHyung Kim, Mirim Ahn, and Taekyoung Kwon. “Location-based key management strong against insider threats in wireless sensor networks”. In: *IEEE Systems Journal* 11.2 (2015), pp. 494–502.
  - [31] Jaewoo Choi, Jihyun Bang, LeeHyung Kim, Mirim Ahn, and Taekyoung Kwon. “Location-based key management strong against insider threats in wireless sensor networks”. In: *IEEE Systems Journal* 11.2 (2017), pp. 494–502.
  - [32] Taehwan Choi, Hrishikesh B Acharya, and Mohamed G Gouda. “The best keying protocol for sensor networks”. In: *Pervasive and Mobile Computing* 9.4 (2013), pp. 564–571.
  - [33] Pete Chown. *Advanced encryption standard (AES) ciphersuites for transport layer security (TLS)*. Tech. rep. RFC 3268, June, 2002.
  - [34] Konstantinos Christidis and Michael Devetsikiotis. “Blockchains and smart contracts for the internet of things”. In: *Ieee Access* 4 (2016).

- 
- [35] Cisco. *Cisco Annual Internet Report (2018–2023) White Paper*. 2020. URL: <https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html>.
- [36] *Contiki*. 2020. URL: <https://github.com/contiki-ng/contiki-ng/wiki>.
- [37] *Contiki Coffee File System*. 2016. URL: [https://anrg.usc.edu/contiki/index.php/Contiki\\_Coffee\\_File\\_System](https://anrg.usc.edu/contiki/index.php/Contiki_Coffee_File_System).
- [38] Whitfield Diffie and Martin Hellman. “New directions in cryptography”. In: *IEEE transactions on Information Theory* 22.6 (1976), pp. 644–654.
- [39] Wenliang Du, Jing Deng, Yungshiang S Han, Pramod K Varshney, Jonathan Katz, and Aram Khalili. “A pairwise key predistribution scheme for wireless sensor networks”. In: *ACM Transactions on Information and System Security (TISSEC)* 8.2 (2005), pp. 228–258.
- [40] Wenliang Du, Jing Deng, Yungshiang Han, Shigang Chen, and Pramod Varshney. “A key management scheme for wireless sensor networks using deployment knowledge”. In: *IEEE INFOCOM 2004*. Vol. 1.
- [41] Xiaojiang Du, Yang Xiao, Mohsen Guizani, and Hsiao-Hwa Chen. “An effective key management scheme for heterogeneous sensor networks”. In: *Ad Hoc Networks* 5.1 (2007), pp. 24–34.
- [42] Adam Dunkels, Joakim Eriksson, Niclas Finne, and Nicolas Tsiftes. *Powertrace: Network-level power profiling for low-power wireless networks*. 2011.
- [43] Donald Eastlake and Paul Jones. *US secure hash algorithm 1 (SHA1)*. 2001.
- [44] Taher ElGamal. “A public key cryptosystem and a signature scheme based on discrete logarithms”. In: *IEEE transactions on information theory* 31.4 (1985), pp. 469–472.
- [45] Mohamed Eltoweissy, M Hossain Heydari, Linda Morales, and I Hal Sudborough. “Combinatorial optimization of group key management”. In: *Journal of Network and Systems Management* 12.1 (2004), pp. 33–50.
- [46] Mohamed Eltoweissy, Mohammed Moharrum, and Ravi Mukkamala. “Dynamic key management in sensor networks”. In: *IEEE Communications magazine* 44.4 (2006), pp. 122–130.
- [47] Mohamed Eltoweissy, Ashraf Wadaa, Stephan Olariu, and Larry Wilson. “Group key management scheme for large-scale sensor networks”. In: *Ad Hoc Networks* 3.5 (2005), pp. 668–688.

- 
- [48] Seyed Hossein Erfani, Hamid HS Javadi, and Amir Masoud Rahmani. “A dynamic key management scheme for dynamic wireless sensor networks”. In: *Security and Communication Networks* 8.6 (2015), pp. 1040–1049.
  - [49] Laurent Eschenauer and Virgil D Gligor. “A key-management scheme for distributed sensor networks”. In: *9th ACM conference on Computer and communications security*. ACM. 2002, pp. 41–47.
  - [50] Toby Gibbs and Suwaree Yordchim. “Thai perception on Litecoin value”. In: *International Journal of Social, Behavioral, Educational, Economic, Business and Industrial Engineering* 8.8 (2014), pp. 2613–5.
  - [51] Eclipse IoT Working Group and IEEE IoT. *IoT Developer Survey Results*. 2019. URL: <https://iot.eclipse.org/community/resources/iot-surveys/assets/iot-comm-adoption-survey-2019.pdf>.
  - [52] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. “Internet of Things (IoT): A vision, architectural elements, and future directions”. In: *Future generation computer systems* 29.7 (2013), pp. 1645–1660.
  - [53] V.C. Gungor and G.P. Hancke. “Industrial wireless sensor networks: Challenges, design principles, and technical approaches”. In: *IEEE Transactions on industrial electronics* 56.10 (2009), pp. 4258–4265.
  - [54] Son N Han and Noel Crespi. “Semantic service provisioning for smart objects: Integrating IoT applications into the web”. In: *Future Generation Computer Systems* 76 (2017), pp. 180–197.
  - [55] Richard Harper. *Inside the smart home*. Springer Science & Business Media, 2006.
  - [56] Xiaobing He, Michael Niedermeier, and Hermann De Meer. “Dynamic key management in wireless sensor networks: A survey”. In: *Journal of Network and Computer Applications* 36.2 (2013), pp. 611–622.
  - [57] Wendi Rabiner Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. “Energy-efficient communication protocol for wireless microsensor networks”. In: *Proceedings of the 33rd annual Hawaii international conference on system sciences*. IEEE. 2000, 10–pp.
  - [58] ASM Sanwar Hosen, Gi-hwan Cho, et al. “A robust key management scheme based on node hierarchy for wireless sensor networks”. In: *International conference on computational science and its applications*. Springer. 2014, pp. 315–329.

- 
- [59] R Housley. *Triple-DES and RC2 key wrapping*. Tech. rep. RFC 3217, December, 2001.
  - [60] Dijiang Huang, Manish Mehta, Deep Medhi, and Lein Harn. “Location-aware key management scheme for wireless sensor networks”. In: *Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks*. ACM. 2004, pp. 29–42.
  - [61] Jyh-Ming Huang, Shun-Bo Yang, and Chan-Ling Dai. “An efficient key management scheme for data-centric storage wireless sensor networks”. In: *IERI Procedia* 4 (2013), pp. 25–31.
  - [62] Junbeom Hur and Hyunsoo Yoon. “A decentralized multi-group key management scheme”. In: *IEICE transactions on communications* 92.2 (2009), pp. 632–635.
  - [63] Ashraf William Hussein Harb and Omayma A. El-Mohsen. “Context aware group key management model for internet of things”. In: *The Seventeenth International Conference on Networks* 28-34 (2018).
  - [64] Tendermint Inc. *Tendermint*. 2020. URL: <https://docs.tendermint.com/master/>.
  - [65] Md Mahidul Islam, Md Zahid Hasan, and Rifat Ali Shaon. “A Novel Approach for Client Side Encryption in Cloud Computing”. In: *2019 International Conference on Electrical, Computer and Communication Engineering (ECCE)*. IEEE. 2019, pp. 1–6.
  - [66] Xiaolin Jia, Quanyuan Feng, Taihua Fan, and Quanshui Lei. “RFID technology and its applications in Internet of Things (IoT)”. In: *2012 2nd international conference on consumer electronics, communications and networks (CECNet)*. IEEE. 2012, pp. 1282–1285.
  - [67] Mohamed Ali Kandi, Djamel Eddine Kouicem, Hicham Lakhlef, Abdelmadjid Bouabdallah, and Yacine Challal. “A Blockchain-based Key Management Protocol for Secure Device-to-Device Communication in the IoT”. In: *proceedings of the 19th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)/ International Workshop on Cyberspace Security (IWCSS)*. IEEE. 2020.
  - [68] Mohamed Ali Kandi, Hicham Lakhlef, Abdelmadjid Bouabdallah, and Yacine Challal. “A Key Management Protocol for Secure Device-to-Device Communication in the Internet of Things”. In: *2019 IEEE Global Communications Conference (Globecom2019)*. Waikoloa, USA, Dec. 2019.



- 
- [69] Mohamed Ali Kandi, Hicham Lakhlef, Abdelmadjid Bouabdallah, and Yacine Challal. “A versatile Key Management protocol for secure Group and Device-to-Device Communication in the Internet of Things”. In: *Journal of Network and Computer Applications* 150 (2020), p. 102480.
  - [70] Mohamed Ali Kandi, Hicham Lakhlef, Abdelmadjid Bouabdallah, and Yacine Challal. “An Efficient Multi-Group Key Management Protocol for Heterogeneous IoT Devices”. In: *IEEE Wireless Communications and Networking Conference (WCNC)*. 2019.
  - [71] Mohamed Ali Kandi, Hicham Lakhlef, Abdelmadjid Bouabdallah, and Yacine Challal. “An Efficient Multi-Group Key Management Protocol for Internet of Things”. In: *26th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. IEEE. 2018, pp. 1–6.
  - [72] Gerd Kortuem, Fahim Kawsar, Vasughi Sundramoorthy, and Daniel Fitton. “Smart objects as building blocks for the internet of things”. In: *IEEE Internet Computing* 14.1 (2009), pp. 44–51.
  - [73] David W Kravitz. *Digital signature algorithm*. US Patent 5,231,668. July 1993.
  - [74] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. *HMAC: Keyed-hashing for message authentication*. 1997.
  - [75] Yi-Hsuan Kung and Hsu-Chun Hsiao. “GroupIt: Lightweight group key management for dynamic IoT environments”. In: *IEEE Internet of Things Journal* 5.6 (2018), pp. 5155–5165.
  - [76] Jae Kwon. “Tendermint: Consensus without mining”. In: *Draft v. 0.6, fall* 1.11 (2014).
  - [77] Tiana Laurence. *Blockchain for dummies*. John Wiley & Sons, 2019.
  - [78] Ao Lei, Haitham Cruickshank, Yue Cao, Philip Asuquo, Chibueze P Anyigor Ogah, and Zhili Sun. “Blockchain-based dynamic key management for heterogeneous intelligent transportation systems”. In: *IEEE Internet of Things Journal* 4.6 (2017), pp. 1832–1843.
  - [79] Ao Lei, Chibueze Ogah, Philip Asuquo, Haitham Cruickshank, and Zhili Sun. “A secure key management scheme for heterogeneous secure vehicular communication systems”. In: *ZTE Communications* 21 (2016), p. 1.
  - [80] Xiaozhou Steve Li, Yang Richard Yang, Mohamed G Gouda, and Simon S Lam. “Batch rekeying for secure group communications”. In: *group* 1 (2001), p. 9.

- 
- [81] Donggang Liu and Peng Ning. “Improving key predistribution with deployment knowledge in static sensor networks”. In: *ACM Transactions on Sensor Networks (TOSN)* 1.2 (2005), pp. 204–239.
  - [82] Z. Liu, X. Huang, Z. Hu, M.K. Khan, H. Seo, and L. Zhou. “On emerging family of elliptic curves to secure internet of things: ECC comes of age”. In: *IEEE Transactions on Dependable and Secure Computing* 14.3 (2017), pp. 237–248.
  - [83] Kejie Lu, Yi Qian, Mohsen Guizani, and Hsiao-Hwa Chen. “A framework for a distributed key management scheme in heterogeneous wireless sensor networks”. In: *IEEE transactions on wireless communications* 7.2 (2008), pp. 639–647.
  - [84] Mingxin Ma, Guozhen Shi, and Fenghua Li. “Privacy-Oriented Blockchain-based Distributed Key Management Architecture for Hierarchical Access Control in the IoT Scenario”. In: *IEEE Access* 7 (2019), pp. 34045–34059.
  - [85] C Madson and N Doraswamy. *The ESP DES-CBC cipher algorithm with explicit IV*. Tech. rep. RFC 2405, november, 1998.
  - [86] Dindayal Mahto and Dilip Kumar Yadav. “RSA and ECC: a comparative analysis”. In: *International journal of applied engineering research* 12.19 (2017), pp. 9053–9061.
  - [87] Dieynaba Mall, Karim Konaté, and Al-Sakib Khan Pathan. “ECL-EKM: An enhanced Certificateless Effective Key Management protocol for dynamic WSN”. In: *International Conference on Networking, Systems and Security (NSysS), 2017*. IEEE. 2017, pp. 150–155.
  - [88] Troy McMillan. *CCNA security study guide: exam 210-260*. John Wiley & Sons, 2018.
  - [89] Mohamed-Lamine Messai, Hamida Seba, and Makhoulf Aliouat. “A new hierarchical key management scheme for secure clustering in wireless sensor networks”. In: *International conference on wired/wireless internet communication*. Springer. 2015, pp. 411–424.
  - [90] *Micaz Wireless Measurement System*. URL: [http://www.openautomation.net/uploads/productos/micaz\\_datasheet.pdf](http://www.openautomation.net/uploads/productos/micaz_datasheet.pdf).
  - [91] *MSP430F5438 Experimenter Board*. 2013. URL: <https://www.ti.com/tool/MSP-EXP430F5438#descriptionArea>.

- 
- [92] Rajasekhar Mungara, K Venkateswararao, and VenkataSubbaReddy Pallamreddy. “A routing-driven elliptic curve cryptography based key management scheme for heterogeneous sensor networks”. In: *Int J Comput Technol Appl* 2.5 (2011), pp. 1690–1696.
- [93] Satoshi Nakamoto et al. “Bitcoin: A peer-to-peer electronic cash system”. In: (2008).
- [94] Nist. *Cryptographic key length recommendation*. 2020. URL: <https://www.keylength.com/en/4/>.
- [95] Luis Nóbrega, André Tavares, António Cardoso, and Pedro Gonçalves. “Animal monitoring based on IoT technologies”. In: *2018 IoT Vertical and Topical Summit on Agriculture-Tuscany (IOT Tuscany)*. IEEE. 2018, pp. 1–5.
- [96] T Okamura. “Lightweight Cryptography Applicable to Various IoT Devices”. In: *NEC Technical Journal* (2017), pp. 67–71.
- [97] Mawloud Omar, Imene Belalouache, Samia Amrane, and Bournane Abbache. “Efficient and energy-aware key management framework for dynamic sensor networks”. In: *Computers Electrical Engineering* 72 (2018), pp. 990–1005. ISSN: 0045-7906. DOI: <https://doi.org/10.1016/j.compeleceng.2018.03.009>. URL: <http://www.sciencedirect.com/science/article/pii/S0045790617322802>.
- [98] *OpenMote-cc2538*. 2020. URL: [https://doc.riot-os.org/group\\_\\_boards\\_\\_openmote-cc2538.html](https://doc.riot-os.org/group__boards__openmote-cc2538.html).
- [99] M. Park, Y. Park, H. Jeong, and S. Seo. “Secure multiple multicast services in wireless networks”. In: *IEEE Transactions on Mobile Computing* (2012).
- [100] Se Jin Park, Murali Subramaniam, Seoung Eun Kim, Seunghee Hong, Joo Hyeong Lee, Chan Min Jo, and Youngseob Seo. “Development of the elderly healthcare monitoring system with IoT”. In: *Advances in Human Factors and Ergonomics in Healthcare*. Springer, 2017, pp. 309–315.
- [101] Mittal K Pedhadiya, Rakesh Kumar Jha, and Hetal G Bhatt. “Device to device communication: A survey”. In: *Journal of Network and Computer Applications* 129 (2019), pp. 71–89.
- [102] Thomas R Peltier. *Information security fundamentals*. CRC press, 2013.
- [103] Zhongyuan Qin, Xinshuai Zhang, Kerong Feng, Qunfang Zhang, and Jie Huang. “An efficient identity-based key management scheme for wireless sensor networks using the bloom filter”. In: *Sensors* 14.10 (2014), pp. 17937–17951.

- 
- [104] Musfiq Rahman and Srinivas Sampalli. “An efficient pairwise and group key management protocol for wireless sensor network”. In: *Wireless Personal Communications* 84.3 (2015), pp. 2035–2053.
  - [105] Sk Md Mizanur Rahman and Khalil El-Khatib. “Private key agreement and secure communication for heterogeneous sensor networks”. In: *Journal of Parallel and Distributed Computing* 70.8 (2010), pp. 858–870.
  - [106] Partha Pratim Ray. “A survey of IoT cloud platforms”. In: *Future Computing and Informatics Journal* 1.1-2 (2016), pp. 35–46.
  - [107] Ronald L Rivest, Adi Shamir, and Leonard Adleman. “A method for obtaining digital signatures and public-key cryptosystems”. In: *Communications of the ACM* 21.2 (1978), pp. 120–126.
  - [108] Ronald Rivest and S Dusse. *The MD5 message-digest algorithm*. 1992.
  - [109] Kiki Rizki. *Efficient Group Key Management for Internet of Things*. 2016.
  - [110] Sushmita Ruj, Amiya Nayak, and Ivan Stojmenovic. “Pairwise and triple key distribution in wireless sensor networks with applications”. In: *IEEE Transactions on Computers* 62.11 (2013), pp. 2224–2237.
  - [111] Ozgur Koray Sahingoz. “Large scale wireless sensor networks with multi-level dynamic key management scheme”. In: *Journal of Systems Architecture* 59.9 (2013), pp. 801–807.
  - [112] Fahad Saleh. “Blockchain without waste: Proof-of-stake”. In: *Available at SSRN 3183935* (2020).
  - [113] Hooman Samani and Rongbo Zhu. “Robotic automated external defibrillator ambulance for emergency medical service in smart cities”. In: *IEEE Access* 4 (2016), pp. 268–283.
  - [114] *Sample Data for powertrace using CM5000 motes*. 2015. URL: <https://github.com/sonhan/contiki/tree/master/apps/powertrace-sonhan/sample-data>.
  - [115] Lakshmi Siva Sankar, M Sindhu, and M Sethumadhavan. “Survey of consensus protocols on blockchain applications”. In: *2017 4th International Conference on Advanced Computing and Communication Systems (ICACCS)*. IEEE. 2017, pp. 1–5.
  - [116] Shadi Al-Sarawi, Mohammed Anbar, Kamal Alieyan, and Mahmood Alzubaidi. “Internet of Things (IoT) communication protocols”. In: *2017 8th International conference on information technology (ICIT)*. IEEE. 2017, pp. 685–690.

- 
- [117] Bruce Schneier. “Description of a new variable-length key, 64-bit block cipher (Blowfish)”. In: *International Workshop on Fast Software Encryption*. Springer. 1993, pp. 191–204.
  - [118] Seung-Hyun Seo, Jongho Won, Salmin Sultana, and Elisa Bertino. “Effective key management in dynamic wireless sensor networks”. In: *IEEE Transactions on Information Forensics and Security* 10.2 (2015), pp. 371–383.
  - [119] Jalpa Shah and Biswajit Mishra. “IoT enabled environmental monitoring system for smart cities”. In: *2016 international conference on internet of things and applications (IOTA)*. IEEE. 2016, pp. 383–388.
  - [120] Alan T Sherman and David A McGrew. “Key establishment in large dynamic groups using one-way function trees”. In: *IEEE transactions on Software Engineering* 29.5 (2003), pp. 444–458.
  - [121] S.R. Singh, A.K. Khan, and T.S. Singh. “A New Key Management Scheme for Wireless Senm Networks using an Elliptic Curve”. In: *Indian Journal of Science and Technology* 10.13 (2017).
  - [122] Secure Hash Standard. “Federal Information Processing Standard (FIPS) 180-2”. In: *National Institute of Science and Technology* (2002).
  - [123] Y. Sun and K.R. Liu. “Hierarchical group access control for secure multicast communications”. In: *IEEE/ACM Transactions on Networking* 15.6 (2007), pp. 1514–1526.
  - [124] Yan Sun, Wade Trappe, and KJ Ray Liu. “A scalable multicast key management scheme for heterogeneous wireless networks”. In: *IEEE/ACM Transactions on networking* 12.4 (2004), pp. 653–666.
  - [125] *Telosb Mote Platform*. URL: [http://www.memsic.com/userfiles/files/Datasheets/WSN/telosb\\_datasheet.pdf](http://www.memsic.com/userfiles/files/Datasheets/WSN/telosb_datasheet.pdf).
  - [126] *The Z1 mote*. 2018. URL: <https://github.com/Zolertia/Resources/wiki/The-Z1-mote>.
  - [127] Gnana Kousalya Chella Thevar and G Rohini. “Energy efficient geographical key management scheme for authentication in mobile wireless sensor networks”. In: *Wireless Networks* 23.5 (2017), pp. 1479–1489.
  - [128] Matthias Thoma, Sonja Meyer, Klaus Sperner, Stefan Meissner, and Torsten Braun. “On iot-services: Survey, classification and enterprise integration”. In: *2012 IEEE International Conference on Green Computing and Communications*. IEEE. 2012, pp. 257–260.

- 
- [129] Marco Tiloca and Gianluca Dini. “GREP: A group rekeying protocol based on member join history”. In: *2016 IEEE Symposium on Computers and Communication (ISCC)*. IEEE. 2016, pp. 326–333.
  - [130] I-Chen Tsai, Chia-Mu Yu, Haruo Yokota, and Sy-Yen Kuo. “Key management in Internet of Things via Kronecker product”. In: *2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC)*. IEEE. 2017, pp. 118–124.
  - [131] M Uma and Ganapathi Padmavathi. “A Survey on Various Cyber Attacks and their Classification.” In: *IJ Network Security* 15.5 (2013), pp. 390–396.
  - [132] Pavel Vasin. “Blackcoin’s proof-of-stake protocol v2”. In: *URL: <https://blackcoin.co/blackcoin-pos-protocol-v2-whitepaper.pdf>* 71 (2014).
  - [133] Luca Veltri, Simone Cirani, Stefano Busanelli, and Gianluigi Ferrari. “A novel batch-based group key management protocol applied to the internet of things”. In: *Ad Hoc Networks* 11.8 (2013), pp. 2724–2737.
  - [134] Wattana Viriyasitavat and Danupol Hoonsopon. “Blockchain characteristics and consensus in modern business processes”. In: *Journal of Industrial Information Integration* 13 (2019), pp. 32–39.
  - [135] Debby Wallner, Eric Harder, and Ryan Agee. *Key management for multicast: Issues and architectures*. Tech. rep. 1999.
  - [136] Changsheng Wan. “IBKES: Efficient Identity-Based Key Exchange with Scalability for Wireless Sensor Networks Using Algebraic Signature.” In: *Adhoc & Sensor Wireless Networks* 39 (2017).
  - [137] Jiuru Wang, Haifeng Wang, Xu An Wang, and Yunpeng Cao. “An Authentication Key Agreement Scheme for Heterogeneous Sensor Network Based on Improved Counting Bloom Filter”. In: *10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*. IEEE. 2015, pp. 815–820.
  - [138] Chung Kei Wong, Mohamed Gouda, and Simon S Lam. “Secure group communications using key graphs”. In: *IEEE/ACM transactions on networking* 8.1 (2000), pp. 16–30.
  - [139] Gavin Wood et al. “Ethereum: A secure decentralised generalised transaction ledger”. In: *Ethereum project yellow paper* 151.2014 (2014), pp. 1–32.
  - [140] Mohamed F Younis, Kajaldeep Ghumman, and Mohamed Eltoweissy. “Location-aware combinatorial key management scheme for clustered sensor networks”. In: *IEEE transactions on parallel and distributed systems* 17.8 (2006), pp. 865–882.

- 
- [141] Mohammad Sadegh Yousefpoor and Hamid Barati. “Dynamic key management algorithms in wireless sensor networks: A survey”. In: *Computer Communications* 134 (2019), pp. 52–69.
  - [142] Zhen Yu and Yong Guan. “A robust group-based key management scheme for wireless sensor networks”. In: *IEEE Wireless Communications and Networking Conference, 2005*. Vol. 4. IEEE. 2005, pp. 1915–1920.
  - [143] Furui Zhan and Nianmin Yao. “A collusion-resistant dynamic key management scheme for WSNs”. In: *Security and Communication Networks* 9.18 (2016), pp. 6351–6364.
  - [144] Furui Zhan, Nianmin Yao, Zhenguo Gao, and Guozhen Tan. “A novel key generation method for wireless sensor networks based on system of equations”. In: *Journal of Network and Computer Applications* 82 (2017), pp. 114–127.
  - [145] Jianmin Zhang, Hua Li, and Jian Li. “Key establishment scheme for wireless sensor networks based on polynomial and random key predistribution scheme”. In: *Ad Hoc Networks* 71 (2018), pp. 68–77.
  - [146] Junqi Zhang and Vijay Varadharajan. “Wireless sensor network key management survey and taxonomy”. In: *Journal of network and computer applications* 33.2 (2010), pp. 63–75.
  - [147] Q. Zhang and Y. Wang. “A centralized key management scheme for hierarchical access control”. In: *Global Telecommunications Conference, 2004. GLOBECOM'04. IEEE*. Vol. 4. 2004, pp. 2067–2071.
  - [148] Yiying Zhang, Xiangzhen Li, Jianming Liu, Jucheng Yang, and Baojiang Cui. “A secure hierarchical key management scheme in wireless sensor network”. In: *international journal of distributed sensor networks* 8.9 (2012), p. 547471.
  - [149] Qian Zhu, Ruicong Wang, Qi Chen, Yan Liu, and Weijun Qin. “Iot gateway: Bridging wireless sensor networks into internet of things”. In: *2010 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*. Ieee. 2010, pp. 347–352.
  - [150] Sencun Zhu, Sanjeev Setia, and Sushil Jajodia. “LEAP+ Efficient security mechanisms for large-scale distributed sensor networks”. In: *ACM Transactions on Sensor Networks (TOSN)* 2.4 (2006), pp. 500–528.

