



**HAL**  
open science

# Blockchain and access control: towards a more secure Internet of Things

Sophie Dramé-Maigné

► **To cite this version:**

Sophie Dramé-Maigné. Blockchain and access control : towards a more secure Internet of Things. Networking and Internet Architecture [cs.NI]. Université Paris Saclay (COmUE), 2019. English. NNT : 2019SACLL018 . tel-02966470

**HAL Id: tel-02966470**

**<https://theses.hal.science/tel-02966470v1>**

Submitted on 14 Oct 2020

**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.

# Blockchain et contrôle d'accès : Vers un Internet des Objets plus sécurisé

Thèse de doctorat de l'Université Paris-Saclay  
préparée à Télécom SudParis

École doctorale n°580 : Sciences et Technologies de  
l'Information et de la Communication (STIC)  
Spécialité de doctorat: Informatique

Thèse présentée et soutenue à Évry, le 12 novembre 2019, par

**Sophie Dramé-Maigné**

Composition du Jury :

Joaquin Garcia-Alfaro Professeur, Télécom SudParis (– SAMOVAR)	Président
Samia Bouzefrane Professeure, CNAM (– CEDRIC)	Rapporteuse
Pascal Lafourcade Maître de Conférence, Université Clermont Auvergne (– LIMOS)	Rapporteur
Hella Kaffel-Ben Ayed Maître de Conférence, Faculté des Sciences de Tunis (– CRISTAL)	Examinatrice
Maryline Laurent Professeure, Télécom SudParis (– SAMOVAR)	Directrice de thèse
Laurent Castillo Ingénieur, Thalès DIS (– Innovation Labs)	Encadrant
Michel Béziat Ingénieur, Thalès DIS (– Innovation Labs)	Invité



Thèse n° 2019SACLL018

THÈSE DE DOCTORAT  
DE L'UNIVERSITÉ PARIS-SACLAY,  
préparée à TELECOM SUDPARIS

École Doctorale N° 580  
Sciences et Technologies de l'Information et de la Communication

Spécialité : Informatique

Par

**Sophie Dramé-Maigné**

**Blockchain et contrôle d'accès : Vers un  
Internet des Objets plus sécurisé**

Thèse présentée et soutenue à Évry, le 12 novembre 2019

Composition du jury :

Joaquin Garcia-Alfaro	Professeur à Télécom SudParis	Président
Samia Bouzefrane	Professeure au CNAM	Rapporteuse
Pascal Lafourcade	Maître de Conférences à l'Université Clermont Auvergne	Rapporteur
Hella Kaffel-Ben Ayed	Maître de Conférences à la Faculté des Sciences de Tunis	Examinatrice
Maryline Laurent	Professeur à Télécom SudParis	Directrice de thèse
Laurent Castillo	Ingénieur chez Thales DIS	Encadrant
Michel Beziat	Ingénieur chez Thales DIS	Encadrant

Mis en page avec la classe thloria-tsp-en adaptée pour Télécom SudParis.



Thèse n° 2019SACLL018

DOCTORAL THESIS  
OF UNIVERSITY PARIS-SACLAY,  
prepared at TELECOM SUDPARIS

Doctoral School N° 580  
Sciences of Information Technology and Communication

Specialty : Computer Science

By

Sophie Dramé-Maigné

**Blockchain and access control:  
Towards a more secure  
Internet of Things**

Thesis presented and defended at Évry, on November, 12th 2019

**Jury Composition :**

Joaquin Garcia-Alfaro	Professor at Télécom SudParis	President
Samia Bouzefrane	Professor at CNAM	Reviewer
Pascal Lafourcade	Associate Professor at Université Clermont Auvergne	Reviewer
Hella Kaffel-Ben Ayed	Associate Professor at Faculté des Sciences de Tunis	Examiner
Maryline Laurent	Professor at Télécom SudParis	Director
Laurent Castillo	Thales DIS Engineer	Advisor
Michel Beziat	Thales DIS Engineer	Advisor

Mis en page avec la classe thloria-tsp-en adaptée pour Télécom SudParis.

## Acknowledgment

First, I would like to thank the wonderful Maryline LAURENT, that embarked on this journey with me after a single meeting and has been the one constant in a sea of changing tides. Thank you for not allowing me to drift, thank you for the advice, the encouragements, and the vote of confidence. I am lucky to have a mentor such as yourself.

Thank you to Samia BOUZEFRANE and Pascal LAFOURCADE for reviewing this manuscript. Many thanks to Joaquin GARCIA-ALFARO and Hella KAFFEL-BEN AYED for accepting to be part of my jury.

A thousand thanks to Hervé GANEM that made all of this possible. He took a chance on me as an intern, and spearheaded a great project that got me hooked enough to want to follow him as a PhD student. I am sorry that we could not finish this venture together but I am forever grateful for the way that it started.

Thank you to Thalès DIS (but really Gemalto) for financing this PhD and welcoming me into its midst. In particular, I would like to thank Laurent CASTILLO, Jerome d'ANNOVILLE, and Michel BÉZIAT for their guidance and understanding. Thank you to Sridharan SADAGOPAN, Sin Jin WONG, Prince, and Olivier VIGNARD for their work on OATL. Thank you also to all of my Gemalto colleagues for the meals shared, the laughs had, and the political views contested.

Thank you to IRT SystemX for setting this thesis in motion. In particular, I am grateful to Philippe WOLF for introducing me to Maryline. Thank you to all of my SystemX colleagues, except for the obvious one, for being the ray of sunshine I needed during difficult times. Thank you for the music we played, the illegal workouts, and the oh so sweet tea.

Thank you to my office mates in TSP that always welcome me with a smile even when they have not seen me in ages. Thank you in particular to my academic siblings: Nesrine and Aymen, for all of their wisdom, and Franklin for the long brain twisting conversations (and for the rides). Thank you also to my fellow PhD student representatives and in particular Abdallah, for sharing in the difficulties of PhD life.

Merci à mes amis qui m'ont accompagnée et soutenue de rebondissement en rebondissement, qui ont vu les hauts et les bas, qui acceptent quand je disparaissais et sont là pour moi quand je refais surface. Thank you in particular to my Cornell friends that inspired me to do a PhD in the first place and work with me to make friendship work across continents and time zones.

Je voudrais remercier mes grands parents pour leur amour d'apprendre, qu'ils ont su transmettre à travers les générations, pour que leur enfants puis leurs petits enfants aient la chance de faire les études qu'ils aimaient tant et ont dû arrêter si jeune. Augustine et Théodore MAIGNÉ ne sont plus parmi nous mais leur héritage et leur influence demeurent.

Merci à ma mère pour tous ses sacrifices, pour ses conseils, et ses encouragements. Merci à Patrick d'être un père sur lequel je peux compter. Merci à ma sœur d'avoir essuyé les plâtres. Merci à toute ma famille petite et grande.

Merci à ceux qui auront trouvé le temps de venir jusqu'à Évry. Merci à ceux qui n'auront pas pu se déplacer.

Merci enfin à Alexandre. Pour tout et pour toujours.





*I dedicate this thesis to the wind.*



## Résumé

Appareils électroménagers intelligents, traqueurs d'activités, voitures connectées, etc. L'Internet des Objets (IdO) est en train de devenir partie intégrante de nos vies. Mais les applications de l'IdO vont au-delà des gadgets intelligents. La domotique peut contribuer à réduire notre impact environnemental. Les réseaux électriques intelligents promettent une meilleure intégration des sources d'énergies renouvelables. Les appareils de télé-santé assurent un meilleur suivi des patients à un coût réduit. Les villes intelligentes pourraient changer notre façon de construire et de vivre la ville. Les appareils de l'IdO peuvent aider à prédire les opérations de maintenance, et, ce faisant, à économiser de l'argent aux entreprises créant par la même occasion un environnement de travail plus sûr.

Les *objets* de l'IdO peuvent prendre plusieurs formes. Fondamentalement, ce sont des objets physiques équipés de capteurs, de logiciels, et autres équipements électroniques qui échangent des données entre eux, ou permettent aux utilisateurs d'agir sur l'environnement voisin de l'objet. Contraints par nature, les appareils de l'IdO manquent de mémoire et de puissance de calcul. Ils sont déployés dans des emplacements potentiellement difficiles à atteindre et jouissent d'un accès au réseau limité. Ils couvrent des cas d'usages variés aux besoins variés et utilisent des protocoles variés. Cela pose de nombreux défis tels que la planification des opérations de maintenance ou la fédération de différents systèmes. Mais le défi le plus important dans l'IdO aujourd'hui est sans doute la sécurité.

La sécurité de l'IdO est l'affaire de tous. Depuis les patients équipés d'un pacemaker connecté, jusqu'aux familles vivant dans les villes intelligentes, même jusqu'à ceux de l'autre côté de la planète qui n'utilisent pas cette technologie. Les risques de sécurité de l'IdO menacent la sécurité des utilisateurs, la vie privée de tous, et représentent des ressources potentielles pour des acteurs malveillants. Une fois compromises, ces ressources peuvent être utilisées dans des attaques à large échelle contre d'autres systèmes tels que l'interruption de services de sites web populaires.

Cette thèse examine et propose des solutions de contrôle d'accès pour l'IdO. Restreindre l'accès aux informations confidentielles et aux fonctions sensibles est la première étape vers la sécurisation d'un système. Les systèmes de contrôle d'accès dictent les règles régissant l'accès et les moyens par lesquels il peut être obtenu. Nous proposons quatre contributions, dont trois liées au contrôle d'accès.

Tout d'abord, nous examinons l'état de l'art pour déterminer l'influence de l'architecture sur les propriétés d'une solution et nous proposons une taxonomie du contrôle d'accès dans l'IdO en fonction de l'architecture.

Ensuite, nous proposons un ensemble de bibliothèques pour aider les développeurs à intégrer les mécanismes de contrôle d'accès dans leurs produits. Ces bibliothèques gèrent l'émission, le stockage, et la vérification de jetons d'autorisation. Ces jetons matérialisent l'autorisation d'accès et sont transportés d'une entité centrale située dans le cloud jusqu'à l'objet connecté via une application mobile sur le téléphone de l'utilisateur.

La blockchain est un registre distribué qui enregistre et ordonne les transactions. Elle est utilisée dans notre troisième contribution pour gérer les attributs des utilisateurs, les politiques de contrôle d'accès, et la confiance entre les différentes entités. Ce faisant, elle fournit un système de contrôle d'accès flexible

et décentralisé dans lequel les décisions sont prises directement par l'appareil IdO.

Notre quatrième contribution élargit le spectre de la sécurité de l'IdO et se concentre sur la propriété des appareils. Au cours de sa vie, un appareil IdO est susceptible d'être revendu ou affecté à différents projets ou gestionnaires au sein d'une même entreprise. Nous fournissons un système de suivi des propriétaires d'un appareil ainsi qu'une preuve de propriété indépendante qui peut être utilisée lors de la revente de l'appareil. De plus, nous proposons un système de gestion des secrets liés aux appareils et un mécanisme de publication des propriétés dynamiques des appareils qui pourraient intéresser des utilisateurs potentiels.

**Mots-clés:** Internet des Objets, IdO, contrôle d'accès, blockchain, propriété, attributs, capacités, jetons.

## Abstract

Smart appliances, fitness trackers, connected cars, etc. The Internet of Things (IoT) is rapidly becoming part of our everyday life. But IoT applications go beyond smart gadgets. Smart homes and smart buildings can help reduce our environmental impact. Smart grids promise to better integrate renewable energy sources. eHealth devices provide a better monitoring and lower patient's costs. Smart cities might reshape the way we build and live. IoT devices can help predict maintenance operation, thus saving the company money and creating a safer work environment in the process.

The *things* of the IoT can take many forms. At their core, they are physical objects fitted with sensors, softwares, and other types of electronics that send data to one another or let users interact with the environment surrounding the object. Constrained by nature, IoT devices lack memory and computational power. They are deployed in locations that may be hard to reach and have limited network coverage. They cover varying use cases with varying requirements and use various protocols. This poses a lot of challenges such as the planning of maintenance operations or the federation of different systems. But arguably the most important challenge in the IoT today is security.

IoT security concerns everyone. From the patients with a connected pacemaker to the families that live in a smart city, even to people on the other side of the planet that don't use that technology. The security risks of the IoT threaten the safety of users, the privacy of everyone, and represent potential resources for malicious actors. Once compromised, these resources can be used to mount large scale attacks on other systems such as the disruption of popular websites.

This thesis examines and proposes access control solutions for the IoT. Restricting access to private information and sensitive functionality is the first step towards securing a system. Access control systems dictate the rules governing access and the means by which it can be obtained. We propose four contributions, among which three are related to access control.

First, we survey the state of the art to determine the influence of architecture on the properties of a solution and we propose an architecture-based taxonomy of IoT access control.

Second, we propose a set of libraries to help developers integrate access control mechanisms into their products. These libraries handle the issuance, storing, and verification of authorization tokens. These tokens materialize authorization and carry it from a central entity located in the cloud to the enforcement mechanism on the IoT device, through a mobile application on the user's phone.

The blockchain is a distributed ledger that registers and orders transactions. It is used in our third contribution to manage user's attributes, access control policies, and trust, providing a decentralized and flexible access control system where decisions are taken directly by the IoT device.

Our fourth contribution broadens the specter of IoT security and focuses on device ownership. Throughout its life, an IoT device is likely to be re-sold or affected to different projects or managers within a company. We provide an ownership tracking system as well as an independent proof of ownership that can be used when re-selling one's device. Additionally, we propose a management system for device-related secret and a mechanism for publishing dynamic device properties that might interest potential users.

**Keywords:** Internet of Things, IoT, access control, blockchain, ownership, attribute-based access control, capability-based access control, tokens.



# Contents

<b>Chapter 1 Introduction</b>	<b>1</b>
1.1 The Internet of Things (IoT)	1
1.2 IoT and challenges	4
1.3 IoT and security	6
1.4 Problem Statement	8
1.5 Contributions	10
1.6 Organization	11
<b>Chapter 2 Background: The Blockchain</b>	<b>13</b>
2.1 Introduction	15
2.2 Cryptographic primitives	16
2.3 Basic Blockchain Concepts	21
2.4 Formal definition	26
2.5 Blockchain Implementations	26
2.6 Consensus protocols	33
2.7 Governance	41
2.8 Blockchain security: attacks	46
2.9 Blockchain security: limitations	54
2.10 Blockchain & IoT	58
2.11 Conclusion	61
<b>Chapter 3 Survey : Access control in the IoT</b>	<b>63</b>
3.1 Introduction	64
3.2 Background	66
3.3 Comparison criteria	73

3.4	Centralized architecture . . . . .	76
3.5	Hierarchical Architecture . . . . .	82
3.6	Federated Architecture . . . . .	85
3.7	Distributed Architecture . . . . .	90
3.8	Taxonomy & Analysis . . . . .	101
3.9	Conclusion . . . . .	104
<b>Chapter 4 OATL: Offline Authorization Token Libraries</b>		<b>105</b>
4.1	Introduction . . . . .	106
4.2	Architecture . . . . .	108
4.3	Tokens . . . . .	112
4.4	Access Control Process . . . . .	117
4.5	Security analysis . . . . .	120
4.6	Proof of Concept (PoC) . . . . .	133
4.7	Comparison to the state of the art . . . . .	134
4.8	Conclusion . . . . .	136
<b>Chapter 5 MAAC-B: Multi-endorsed Attributes for Access Control using the Blockchain</b>		<b>141</b>
5.1	Introduction . . . . .	142
5.2	Architecture, Assumptions and Overview . . . . .	144
5.3	Trust Anchor Service . . . . .	148
5.4	Policy Service . . . . .	149
5.5	Attribute Service . . . . .	153
5.6	Access Control . . . . .	155
5.7	Security Analysis . . . . .	158
5.8	Comparison to the state of the art . . . . .	161
5.9	Conclusion . . . . .	162
<b>Chapter 6 IoT devices' lifecycle: ownership change and remote configuration</b>		<b>165</b>
6.1	Introduction . . . . .	166
6.2	Security Considerations . . . . .	167
6.3	Asset ownership . . . . .	169



---

6.4	Managing secrets . . . . .	176
6.5	Sharing configurations . . . . .	180
6.6	Conclusion . . . . .	182
<b>Chapter 7 Conclusion and Perspectives</b>		<b>185</b>
7.1	Conclusion . . . . .	185
7.2	Perspectives . . . . .	192
<b>Appendices</b>		<b>197</b>
<b>Appendix A Publication List</b>		<b>199</b>
A.1	Publications in journals, Conferences and Workshops . . . . .	199
A.2	Work under review . . . . .	199
<b>Appendix B Table of acronyms</b>		<b>201</b>
<b>Appendix C Blockchain et controle d'accès : Vers un Internet des Objets plus sécurisé -</b>		
<b>Résumé</b>		<b>205</b>
C.1	Introduction . . . . .	205
C.2	Survol du contrôle d'accès dans l'IdO . . . . .	207
C.3	Librairies de jetons d'autorisation hors-ligne . . . . .	207
C.4	Contrôle d'accès à base d'attributs à approbations multiples utilisant la blockchain . . . . .	209
C.5	Le cycle de vie des objets de l'IdO : Changement de propriétaire et configuration à distance . . . . .	210
C.6	Conclusion . . . . .	211
<b>Bibliography</b>		<b>213</b>



# List of Figures

1.1	Total number of active device connections worldwide . . . . .	2
1.2	Several types of IoT users . . . . .	4
1.3	The challenges of the IoT . . . . .	5
2.1	Hash function . . . . .	16
2.2	Applying MD5 to different inputs . . . . .	17
2.3	Digital Signature . . . . .	18
2.4	Merkle tree . . . . .	20
2.5	Merkle tree: modifying a data block . . . . .	21
2.6	Merkle tree: proof of membership . . . . .	22
2.7	Overview of the blockchain . . . . .	23
2.8	Blocks . . . . .	24
2.9	Blockchain fork . . . . .	25
2.10	Bitcoin transaction model . . . . .	27
2.11	Logos of various cryptocurrencies, including Doge coin . . . . .	29
2.12	Computing Power distribution in Bitcoin . . . . .	34
2.13	An example of fork attempts with exponential subjective scoring . . . . .	38
2.14	Double Spending . . . . .	47
2.15	Propagation delay leading to a fork . . . . .	55
2.16	Bitcoin Nodes around the world . . . . .	57
2.17	Should you use a blockchain ? . . . . .	60
3.1	Instance of access control request . . . . .	70
3.2	Centralized Architecture . . . . .	76
3.3	Ordered PAP . . . . .	83
3.4	Ordered PDP . . . . .	84
3.5	Federated architecture . . . . .	87
3.6	Access request in the hybrid architecture . . . . .	92
3.7	Multi PDP architecture . . . . .	94
3.8	Blockchain-based architecture . . . . .	97
3.9	A Taxonomy of access control architectures . . . . .	103

4.1	OATL architecture . . . . .	111
4.2	OATL example use case - Renting a car . . . . .	112
4.3	Token #1 - Keys . . . . .	114
4.4	Implicit revocation without time-awareness . . . . .	115
4.5	Explicit revocation without time-awareness . . . . .	116
4.6	Access Request . . . . .	119
4.7	OATL - Assets sensitivity level . . . . .	123
4.8	Threats addressed by OATL . . . . .	128
4.9	OATL implementation architecture . . . . .	133
4.10	Mobile Application GUI . . . . .	135
5.1	Access policy issuance and update . . . . .	146
5.2	Attribute management and evaluation . . . . .	147
5.3	Trust Anchor Contract . . . . .	148
5.4	Policy Contract (PolC) . . . . .	150
5.5	Dispatch Contract . . . . .	151
5.6	AttC: Attribute endorsement and revocation . . . . .	153
5.7	Attribute Retrieval . . . . .	156
6.1	An example of blockchain transactions . . . . .	172
6.2	Overview of ownership transfer . . . . .	173
6.3	Proof of ownership . . . . .	174
6.4	Transferring ownership and delivering device secret . . . . .	178
6.5	Publishing a new secret to the blockchain . . . . .	179
6.6	Publishing dynamic properties to the blockchain . . . . .	181
6.7	Threats addressed by our proposal . . . . .	182

# List of Tables

3.1	Summary of solutions adopting a centralized architecture without serverless authorization	81
3.2	Summary of solutions adopting a centralized architecture with serverless authorization	82
3.3	Summary of solutions adopting a hierarchical architecture	86
3.4	Summary of solutions adopting a federated architecture	89
3.5	Summary of solutions adopting an hybrid distributed architecture	91
3.6	Summary of solutions adopting a distributed architecture with multiple PDP	95
3.7	Summary of solutions adopting a blockchain-based architecture	98
3.8	Comparison of architectures	102
4.1	Notations for OATL	109
4.2	Token types	113
4.3	Comparison of OATL with the state of the art	137
5.1	Functions of an Trust Anchor smart Contract (TrAnC)	149
5.2	Functions of a Policy smart Contract (PolC)	151
5.3	Functions of a Dispatch smart Contract (DisC)	152
5.4	Functions of an Attribute smart Contract (AttC)	154
5.5	Comparison of MAAC-B with the state of the art	163
6.1	Chain of Ownership: Notations	167
6.2	Threats	169
6.3	Transaction format	171
6.4	Input format	172
6.5	Output format	172
7.1	Research questions in contributions	189
B.1	Acronyms	204



# Chapter 1

## Introduction

### Contents

---

<b>1.1</b>	<b>The Internet of Things (IoT)</b> . . . . .	<b>1</b>
1.1.1	Its many applications . . . . .	2
1.1.2	Its many users . . . . .	4
<b>1.2</b>	<b>IoT and challenges</b> . . . . .	<b>4</b>
1.2.1	Resource Efficiency . . . . .	5
1.2.2	Longevity . . . . .	5
1.2.3	Business model . . . . .	6
<b>1.3</b>	<b>IoT and security</b> . . . . .	<b>6</b>
1.3.1	Access control in IoT . . . . .	7
1.3.2	Lifecycle and ownership . . . . .	8
<b>1.4</b>	<b>Problem Statement</b> . . . . .	<b>8</b>
<b>1.5</b>	<b>Contributions</b> . . . . .	<b>10</b>
<b>1.6</b>	<b>Organization</b> . . . . .	<b>11</b>

---

### 1.1 The Internet of Things (IoT)

In 1982, a modified Coke machine was introduced at Carnegie Mellon University. It could report its inventory and comment on the temperature of newly loaded drinks. This machine was the first Internet-connected appliance. The Internet of Things (IoT) truly blossomed in the early 2000s when advances in different technologies such as wireless communications, the Internet, and micro-electromechanical systems finally caught up to its ambitions.

The IoT provides physical objects with sensors, software and electronics, that enable the transmission of data between them or other entities without any human intervention. This integration of the digital in the physical promises to improve the efficiency, accuracy, and economic value of a system. In August

2018, IoT Analytics<sup>1</sup> projected the number of active IoT devices to 21.5 billions in 2025, not including mobile phones, tablets, laptops and the likes. Their statistics are reproduced in Figure 1.1.

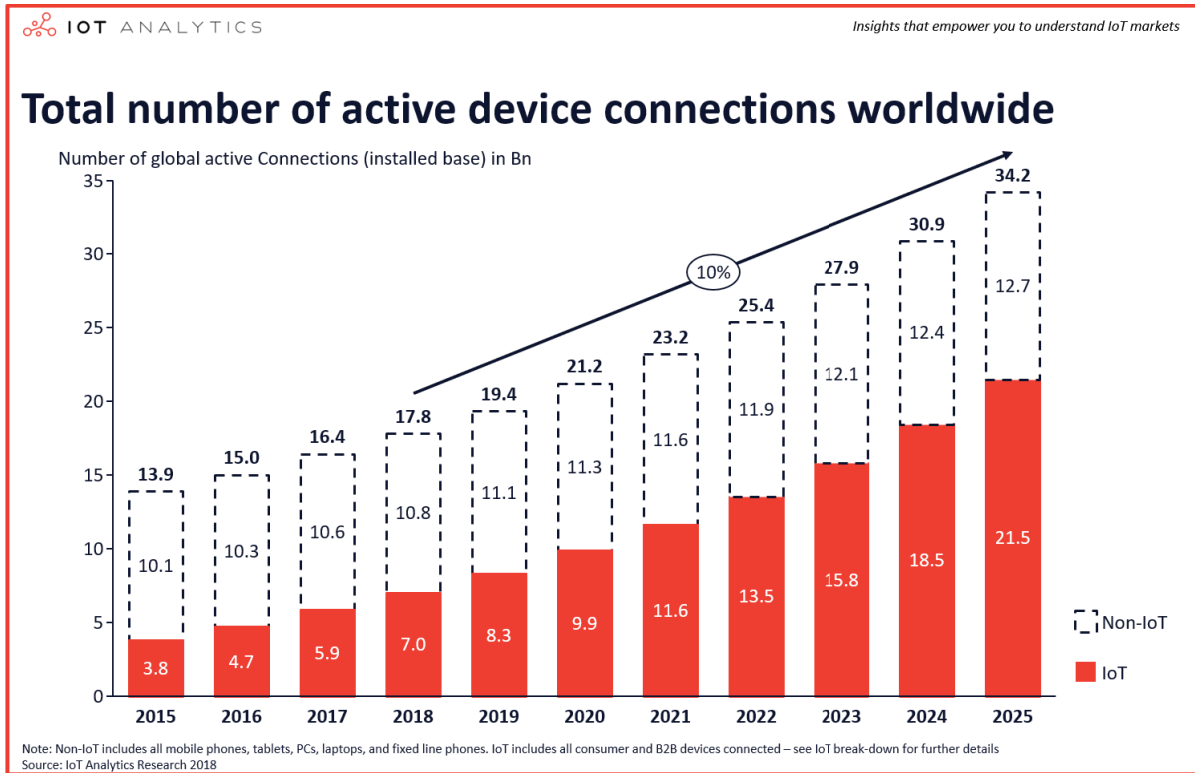


Figure 1.1

### 1.1.1 Its many applications

Application domains for the IoT are numerous and diverse. Below are the six main area of applications.

**Smart Gadgets** The IoT device that we are probably the most familiar with is the fitness tracker: a bracelet that counts our steps, monitors our heartbeats, detects when we are exercising, and sends all that data to our phone to provide us with numerous graphs. There are a number of IoT devices that, like fitness trackers, can be categorized as smart gadgets: connected plushies that kids and parents can use to send each other messages, smart shoes with sensors embedded in the sole, sex toys that can be activated from anywhere in the world, etc. These objects are often used in isolation via a vendor-provided application. Usability is the main requirement for such products.

<sup>1</sup><https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/>, Last checked: July 10th, 2019



**eHealth** A Bluetooth-enabled insulin pump. An at-home kit containing a scale or electrodes that the patient can use themselves, and that send the result directly to their physician. A cloud-connect pacemaker that makes doctor appointment on your behalf and sends data necessary for follow up. The eHealth is composed of connected medical devices, systems that store and manage medical data, and the services that can be built using both. Privacy is a concern for patients but is also often mandated by law.

**Smart Home** There are much improvements that can be gain in our houses from the integration of IoT devices. Convenience is the first example: From a fridge that shops on its own, to lights that turn on and off automatically, or a stove top that turns itself off when users have been away for too long. Smart homes, as they are called, can also help reduce energy consumption by running the washing machine or refilling the battery of your electric car in the middle of the day, as the sun shines on the solar panels on the roof. Sensors can also help monitor air quality, smoke, carbon monoxide levels, etc. Automatic doors and other smart appliances can greatly improve the quality of life for the elderly and disabled. Smart homes are characterized by the inclusion of IoT devices into an overall system where devices can be connected to one another and work in tandem, or at least can be operated from a central interface.

**Smart Building** The equivalent of smart homes for the private sector is smart buildings: offices with blinds that go down automatically when sunshine hits the window, access controlled by smart locks, cameras that can detect potential threats and inform the security officer, etc. If the technology deployed here can be similar to the devices used in smart homes, the scale, the requirements, and the threat model differ in crucial ways. The number of expected users for instance is drastically different. Revocation is a rare occurrence in a smart home where family members are constant while it is fairly common for a company with temporary employees, or within a building that rents out offices to start ups and smaller companies. The energy requirements are much higher, the lifetime of devices is shorter, etc.

**Smart City and infrastructure** Moving up the scale once more, *smart cities* offer yet another landscape for the IoT. Garbage pick up can be optimized to not interfere with the circulation, traffic signals can operate differently throughout the day to accommodate a bigger pedestrian traffic, the city can better identify where new equipment such as public benches or restroom should be deployed, etc. IoT devices can also be deployed over other kind of infrastructure such as forests, alerting the authorities at the first signs of fire, highways, to monitor traffic, accidents, and maintenance operation, or electric grids (*smart grids*), to optimize energy production according to the demand. The latter example would help integrate more renewable energy sources into the network as the existing infrastructure is not built to operate with varying inputs that are inherent to solar panels for instance.

**Industry 4.0** The industrial sector has been leading the automation effort for many decades. The IoT is the next step in this evolution. IoT devices can be used to predict maintenance operation, track inventory along the supply chain, and move decision making closer to the machines. In this sector, device to devices communication is expected to be more prevalent than in user-facing domains such as smart homes. These devices will operate in more sensitive environments where safety and resilience matter most.

### 1.1.2 Its many users

With applications as diverse as those mentioned in the previous section, the IoT is bound to have heterogeneous users. We take a look at what these users might be. In this particular context, *user* will refer to anybody that is affected by a service, either because they use it directly, because it stores information about them, or because it affects their environment.

With that definition in mind, we separate users between voluntary and involuntary users.

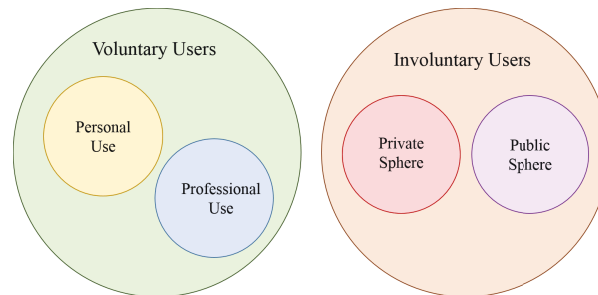


Figure 1.2: Several types of IoT users

**Voluntary Users** These are the people that chose to use IoT device, introduced smart appliances into their house, wear a fitness tracker, etc. They care most about usability. Within this category, there are those who use their device in their personal life and those we use it in the context of work.

Within the industry in particular, the requirements can vary widely depending on the type of activity: manufacturer, service provider, user-facing business, business-to-business, ...

**Involuntary User** By nature of applications such as smart cities and smart buildings, some users may be confronted to IoT devices involuntarily. This affects the private sphere as is happening in France with Linky<sup>2</sup>, the connected electric meter that is being deployed country-wide to the displeasure of many individuals. Users can also be exposed to the IoT in public spaces. London is known for its many security cameras. On a work day, the average London citizen is believed to step in front of around 300 cameras<sup>3</sup> operated by a mixture of private and public entities. These systems generate information that can be of a sensitive nature. Privacy is therefore a big concern.

## 1.2 IoT and challenges

Though they have great potential, IoT solutions face a number of challenges, summarized in Figure 1.3. These challenges are faced while developing an IoT product but also by the IoT sector as a whole that

<sup>2</sup><https://www.fournisseurs-electricite.com/guides/compteur/linky/refuser>, Link in French, Last checked: July 8th, 2019

<sup>3</sup><https://www.cctv.co.uk/how-many-cctv-cameras-are-there-in-london/>, Last Checked: 08/07/2019

must come together to try and solve them. This section dives into the resource efficiency, longevity, and business model issues. Security is the topic of Section 1.3.

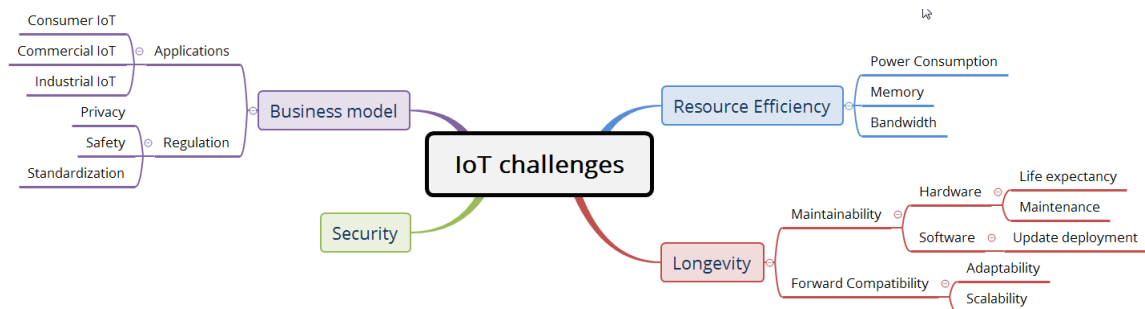


Figure 1.3: The challenges of the IoT

### 1.2.1 Resource Efficiency

The defining challenge of the IoT is its lack of resources: power consumption, memory use, bandwidth, all must be used sparingly. There is a need for tools and protocols that can work in constrained environments. Technological advances can somewhat mitigate this issue.

Apart from the code running on the device itself, larger scale methods can be employed to distribute requests amongst devices, optimize the consumption of resources throughout the network, etc.

Resource usage has a direct influence not only on cost, but also on a device's availability. A solar-powered device for instance only has a finite amount of energy available. After it has been spent, the device will power down until the next ray of sunshine. A device with a full memory will not be able to store sensed data after that point, thus failing to complete its purpose.

### 1.2.2 Longevity

The life-cycle of a system is an issue that must be addressed before its deployment: How long is the system expected to function for? How will it be maintained? To what extent can it be modified? How will the end of life be handled? The maintainability of an IoT solution can be divided into two aspects: hardware and software.

Some IoT systems require the deployment of a number of devices over many locations. In smart grids for instance, sensors must cover the entirety of the electricity grid. They can be installed in hard-to-reach places. It is certainly too costly and too slow to visit each device for a manual maintenance operation. Being deployed in the wild, IoT devices also have to take into account physical threats such as natural disasters or acts of vandalism. The life expectancy and durability of the underlying hardware is therefore an important issue.

When physical maintenance is not an option, one must turn to remote software updates. The availability of the device can then become an issue. Some devices are indeed highly constrained energy-wise and will turn themselves off when not in use. Others simply operate with lower range protocols and

cannot be reached remotely. Devices may become unavailable during the update, which might be unacceptable depending on the device's purpose. Update deployment is therefore a tricky topic in the IoT.

Another important aspect of the longevity of a solution is its forward compatibility. Requirements change, standards evolve, hardware can be re-purposed, etc. IoT solutions should be adaptable. Especially to achieve high life expectancy in the face of high deployment and maintenance costs. Furthermore, the IoT is still an emerging domain. Use cases and user demands are likely to evolve in unpredictable ways over the next decade. Deployed solutions should be able to evolve as well or they will soon become obsolete.

Where traditional solutions operate with static requirements and a low user turnover, some IoT use cases require dynamic user addition, involve mobile targets, and ever-changing requirements. Scalability (up and down) is therefore an important component in IoT applications.

### 1.2.3 Business model

As we have seen in Section 1.1.1, IoT use cases are incredibly diverse. The IoT market can be fragmented in roughly three segments: Consumer IoT, serving individuals, Commercial IoT, for professional use, and Industrial IoT, for industrial applications. Each segment comes with its own set of challenges and requirements. The Industrial IoT in particular, must integrate to an environment made up of very old, proprietary systems that are costly to replace.

With so many pieces in play and so many diverging interests, it can be hard to work toward a common goal to advance the IoT domain as a whole. Nonetheless, standards are slowly starting to emerge. But the landscape remains highly heterogeneous, thus compromising interoperability. Regulations are also emerging to control things like privacy options.

## 1.3 IoT and security

Security is probably the biggest challenge facing the IoT today. The rapid expansion of the field came at the expense of security as vendors rushed to get their product to market. The security model differs from classical system as threats that were mainly digital are moving to the physical world.

The security incidents that routinely hit the IoT are making headlines in mainstream newspapers and eroding user's trust, thus slowing down user adoption. Securing the IoT is a vital step in its future growth.

**Impact on voluntary users** The majority of devices are in the hands of non-tech-savvy users that are notably bad at keeping software up-to-date. When in use, IoT devices might not be connected to the Internet. The challenges in deploying security patches extend the life of IoT vulnerabilities that can be exploited almost forever. And the trend is not receding<sup>4</sup>.

The most immediate consequence is the risk that information transiting through such systems might be stolen or modified. The sensitivity of that information can be obvious, i.e. in eHealth services, but seemingly harmless devices, such as a smart watch, can reveal more than the users might be comfortable

---

<sup>4</sup><https://securelist.com/new-trends-in-the-world-of-iot-threats/87991/>, Last checked: February 26th, 2019

with: location, sleeping pattern, habits, etc.. Devices that can be acted upon (actuators) pose another class of risks, ranging from discomfort to dangerous: The Air Conditioning of a smart building can be made to turn the workplace into a fridge, a home owner can be locked out by his smart lock, a corrupted car<sup>5</sup> can cause a lethal accident on the highway.

**Impact on involuntary and non-users** The lack of user protection in regulations can cause individuals to be forced into using IoT devices: in the US for instance, connected pacemakers can be implanted without patient consent<sup>6</sup>. The security and privacy of such systems becomes even more important than as users cannot opt out of their use. It is also the case for large scale systems that optimize resource use by analyzing user behaviour. In Europe, legislation such as the General Data Protection Regulation (GDPR) has been introduced to protect individuals. However compliance can be technically difficult if these requirements are not taken into account early enough in the development process.

The scope of the threat extends even to non-users: the Mirai Botnet [Antonakakis et al., 2017] used a network of corrupted IoT devices to launch Distributed Denial of Service (DDoS) attacks<sup>7 8</sup> that noticeably disturbed internet traffic on a large scale.

Unprotected IoT devices constitute an extraordinary resource for malicious actors. IoT security is therefore everyone's business.

### 1.3.1 Access control in IoT

Authorized access is a requirement that stands at the heart of any system. Access control (AC), which purpose is to determine who can access what, in what manner, and according to what rules, is therefore an important part of a system's protective arsenal.

There are two concurrent approaches to designing an access control solution for the IoT. One can try and adapt existing solutions, or start anew. Indeed, the IoT paradigm departs from classical systems in significant ways.

As with everything in the IoT, the first issue concerns resources. Most classical access control solutions are performed in a client-server model. In this model, the *objects* are stored on a server that is expected to have a fair amount of resources at its disposal: *memory* to store all the clients information and access rules, maybe even log access requests, *computational power* to run the decision engine and deal with concurrent client requests, *bandwidth* to serve a great number of clients. The server should also be available at all time.

In IoT use cases, the target of access requests are hosted on constrained devices that cannot perform the traditional server role. Additionally, requestors can be device themselves and suffer the same resource constraints. Classical access control solutions therefore cannot be used in their current state, and need

---

<sup>5</sup><https://medium.com/s/new-world-crime/a-brief-history-of-hacking-internet-connected-cars-and-where-we-go-from-here-5c00f3c8825a>, Last checked: March 4th, 2019

<sup>6</sup><https://www.theatlantic.com/technology/archive/2018/01/my-pacemaker-is-tracking-me-from-inside-my-body/551681/>, Last checked: July 8th, 2019

<sup>7</sup><https://www.theguardian.com/technology/2016/oct/26/ddos-attack-dyn-mirai-botnet>, Last checked: March 4th, 2019

<sup>8</sup><https://krebsonsecurity.com/2018/05/study-attack-on-krebsonsecurity-cost-iot-device-owners-323k/>, Last checked: March 4th, 2019

to be adapted or abandoned entirely. These modifications can make legacy solutions cumbersome and ill-fitted.

Sometimes, security simply takes space that the device cannot afford. In this case, access control has to be performed outside of the device, often by a centralized server, which poses its own security issues.

Where to take the access control decision is the subject of another debate dividing the IoT access control community. Centralized systems have the advantages of erasing most of the resource constraint issues and are capable of implementing legacy solutions. They however present a single point of attack that makes a juicy target for attackers. Distributed solutions are harder to deploy but offer more resilience and flexibility. Recently, the blockchain has emerged as a good candidate for distributing access control systems. The expressivity of such system remains limited.

Some IoT use cases, while they allow a bootstrap phase, require access control to only involve local entities. This is the case for instance of a connected car that can be open with one's phone, which should still be possible in a subterranean parking lot with no network coverage. The question of offline access is therefore an alley that needs exploring regardless of architectural choice.

### 1.3.2 Lifecycle and ownership

When re-selling a computer, a phone, or a tablet, most users understand that some steps must be taken to ensure that the new owner will not get access to their data. Consumer IoT devices are as likely to get re-sold as any other appliance. Devices can also be re-purposed within a company and need to be assigned to a new project or administrator.

Not many solutions exist to handle this change in ownership. It is part of the lifecycle of IoT devices all the same. In the academic literature, the transfer of ownership is addressed at the device level [Osaka et al., 2008, Ray et al., 2018]. Ownership transfer is defined [Rekleitis et al., 2014] as “the capability to pass ownership of a tag to a third party without compromising backward untraceability for the said party or forward untraceability for the previous owner.” The focus is on key management and domain boundaries. The devices that are concerned by these protocols are RFID (Radio Frequency Identification) tags. No record is kept of past owners.

Some solutions implement ownership tracking via a blockchain. On the Bitcoin blockchain, Colored Coins [Rosenfeld, 2012] can be used to track asset exchanges. On the Ethereum blockchain [Buterin et al., 2013, Wood, 2014], smart contracts [Szabo, 1997] can be programmed to do similar things. Other blockchains such as NXT<sup>9</sup> provide a native asset exchange. There are also front-end applications<sup>1011</sup> that bridge several blockchains together to facilitate interoperation. These implementations are not IoT-specific but their general-purpose tokens are IoT-compatible.

## 1.4 Problem Statement

From the previous discussion, we extract the following research questions that we try to answer throughout this thesis:

---

<sup>9</sup>NXT: <https://nxtplatform.org>, Last Checked: July 28th, 2019

<sup>10</sup>Melonport: <https://melonport.com>, Last Checked: July 28th, 2019

<sup>11</sup>Exodus: <https://www.exodus.io>, Last Checked: July 28th, 2019

**Question 1** What are the main remaining challenges of access control in the IoT?

We have mentioned a few challenges faced while designing access control solutions for the IoT, namely resource constraints, adaptation of existing solutions, and decentralization. These issues are the ones that arise naturally when discussing the topic. The research community has been tackling them for more than a decade. We wonder how successful that process has been, what challenges were discovered in the process, and where the future of access control solutions for the IoT lies.

**Question 2** What role does architecture play in the existing access control solutions?

Centralization is used to mitigate resource constraints. Distribution is used to mitigate the scalability and security issues associated with centralization. The ideal architecture for access control is still being debated. Definitions of what exactly constitute a centralized or distributed solution vary. Other architectures exist between these two extremes that could benefit the debate and constitute an acceptable compromise. We would like to identify properties, benefits, and disadvantages associated to different architecture to better navigate this issue.

**Question 3** Can access control logic be deported to edge node to enable serverless authorization decisions?

IoT networks are layered. At the edge of the network are the devices themselves. These devices are often associated to a local gateway with more resources that can help manage the device, and relay information to other layers. The local gateway can communicate with a local server that itself communicates with a cloud platform, etc. Information is usually processed at the center of the network, on servers with a lot of storage space, computation power, high network connectivity, etc. But some use cases call for information to be processed on the devices, i.e. on the edge of the network. For various reasons such as low connectivity, high time sensitivity, to lower the amount of data transmitted to or stored in the cloud, the device may be required to make an access control decision without the help of a central server.

**Question 4** How can we increase security usability?

Security is not commonly used not only in the IoT but in the digital world in general. The general public tends to see it as an hindrance. So security solutions are only as useful as they are used. Usability is therefore paramount to the success and adoption rate of a solution. Usability should target two populations. The first is the end-user of a product. But the second is arguably more important: it is IoT device's vendors and developers. Indeed, if the security mechanisms are not implemented in a product, there is no need to convince individuals and companies to use it. So the question is, can we tackle the problem at the design phase by making security integration easier for developers.

**Question 5** Can the blockchain be leveraged for decentralizing access control while maintaining expressivity and offline access?

The blockchain has emerged as a tool to decentralize and distribute trust. It has evolved past cryptocurrencies and is now used for many other applications. IoT devices may not possess the capacities necessary to work in such a demanding ecosystem. We wonder what kind of information can be posted

on the blockchain, the level of flexibility and expressivity that can be maintained through the use of smart contracts, as well as the possibility of operating without a continuous access to the blockchain network.

**Question 6** Using the blockchain, can we tackle other IoT issues such as the lifecycle of devices?

IoT devices hold a peculiar place in the digital space. They have access to very personal data and can be used to infiltrate private networks. But they are also everyday objects that are unassuming to users. The security paradigm is fundamentally different than with servers, laptops, or even phones. Indeed, IoT devices are bought or sold second-hand, given to friends and family, discarded without care. In the industry, the volume of expected devices is dizzying. Similarly, they are bound to change hands, be affected to different projects or locations, as tend to happen with company assets. Their ownership status and lifecycle is therefore an important subject that the blockchain could help tackle. The possibilities offered by the blockchain are plentiful. Asset transfer is one of its core applications. Can this be used to facilitate the management of devices' lifecycle ?

## 1.5 Contributions

This thesis is made up of four distinct contributions, three of which pertain to access control.

**Contribution 1: Survey of access control solution for the IoT** Our first contribution is a survey on existing access control solutions for the IoT. We define classical access control architectures using the four core functions of access control (PEP, PDP, PAP, PIP). The strengths and weaknesses of each architecture are discussed in details. Solutions from the literature are provided to illustrate each architecture. Each reference is analyzed in details and evaluated on objective and qualitative criteria. An architecture-based taxonomy of IoT access control is proposed. Finally, potential future research direction are presented.

This contribution answers questions 1, 2, 3, and 5.

**Contribution 2: Token libraries for serverless access control** Our second contribution is a set of libraries used to issue, store, and interpret authorization tokens. They are aimed at developers and focus on usability and ease of integration. Each library is to be used by a different actor: a cloud platform, a mobile application, and an IoT device. The solution is serverless with local conditions evaluated on the device at access time. It is also designed for flexibility and modularity. Several types of tokens are proposed to accommodate the capacities of different devices as well as different use cases. We offer a thorough security analysis of the proposal as well as a Proof of Concept implementation illustrating the high usability of our solutions for developers and users alike. A live coding video demonstration has been shot to showcase the ease of integration of our libraries.

This contribution answers questions 3 and 4.

**Contribution 3: Attribute-based access control over the blockchain** The third contribution presents a second IoT access control solution. It is based on attributes that are stored on the blockchain and can be endorsed by any willing entity. User identities are blockchain-based and user-controlled. They can be used to separate attributes in accordance with each user's wishes. Our attribute endorsement system is



independent of any single entity. As such, it resists operational changes such as the disappearance of a company, or the introduction of new actors in the ecosystem. The blockchain additionally hosts a trusted entities management system enabling administrative changes throughout the life of IoT devices, and a distributed policy management system with generic policies based on attributes. The trust level of an attribute is computed based on the reputation of the entities that endorsed it. This trust level is used to parametrize access control policies. This defines an overall user-centric access control system enabling interoperability and flexibility.

This contribution answers questions 3 and 5.

**Contribution 4: Blockchain-based transfer of ownership** Our last contribution departs from access control and takes a look at IoT device ownership. It defines an independent proof of ownership based on blockchain transactions, desintermediates and decentralizes ownership records. A key management system is proposed to handle the transmission of the security context from one owner to the next. Finally, the same system can be used to advertise dynamic device properties to potential users.

This contribution answers questions 4 and 6.

## 1.6 Organization

Definition of acronyms can be found in Table B.1, in Annex. The remainder of this thesis is organized as follows:

**Chapter 2** introduces the blockchain: the underlying cryptographic primitives, its basic concepts, the main security issues, limitations both in its functioning and security-wise. The adequacy of the blockchain with IoT application is questioned. The purpose of this initial chapter is to get all the blockchain questions out of the way, acquainting the reader with a concept that is unfamiliar to many. The blockchain will be used as a tool in latter chapters. Readers should therefore be aware of its strength and shortcomings.

**Chapter 3** surveys access control solutions in the IoT. Their architecture is used to classify them: centralized, distributed, but also hierarchical and federated architectures are defined, examined, illustrated, and compared. This chapter provides an extensive review of the state of the art as well as future research directions in the field. This chapter presents our first contribution.

**Chapter 4** presents our token-based access control libraries. This project sets out to provide IoT developers with pre-packaged access control to increase its usability and therefore adoption. Our solution enables offline access. Two types of token are presented to accommodate many IoT use cases. One uses symmetric cryptography, the other uses asymmetric keys. Our Proof of Concept implementation is presented as well as improvements for a second version of the libraries. This chapter presents our second contribution.

**Chapter 5** uses the blockchain to distribute attribute endorsements. The chapter discusses the merits of Attribute-Based Access Control. The issuance and management of attributes is presented as well as the management of access control policies. A detailed security analysis is provided to validate our proposal. This chapter presents our third contribution.

**Chapter 6** departs from access control and focuses on ownership records. The blockchain is used to track ownership changes, and provide independent proofs of ownership. An extension allows users to manage device's credentials, while another lets the owner share its device configuration for potential device to device interactions. This chapter presents our fourth contribution.

**Chapter 7** concludes this thesis. We summarize our journey and reflect on its success at answering the research questions presented in the previous section. Perspectives for the future are broached.

# Chapter 2

## Background: The Blockchain

### Contents

---

<b>2.1</b>	<b>Introduction</b> . . . . .	<b>15</b>
<b>2.2</b>	<b>Cryptographic primitives</b> . . . . .	<b>16</b>
2.2.1	Cryptographic hash functions . . . . .	16
2.2.2	Digital Signatures . . . . .	18
2.2.3	Merkle trees . . . . .	19
<b>2.3</b>	<b>Basic Blockchain Concepts</b> . . . . .	<b>21</b>
2.3.1	Transactions . . . . .	22
2.3.2	Addresses . . . . .	22
2.3.3	Miners . . . . .	22
2.3.4	Blocks . . . . .	23
2.3.5	Smart Contracts . . . . .	24
2.3.6	Forks & Longest Chains . . . . .	25
<b>2.4</b>	<b>Formal definition</b> . . . . .	<b>26</b>
<b>2.5</b>	<b>Blockchain Implementations</b> . . . . .	<b>26</b>
2.5.1	Bitcoin . . . . .	27
2.5.1.1	Transaction-based model . . . . .	27
2.5.1.2	Wallets . . . . .	28
2.5.1.3	Scripts . . . . .	28
2.5.1.4	Alt-coins . . . . .	28
2.5.2	Ethereum . . . . .	29
2.5.2.1	Accounts . . . . .	29
2.5.2.2	Messages and Transactions . . . . .	29
2.5.2.3	Crypto-fuel . . . . .	30
2.5.2.4	The Ethereum Virtual Machine (EVM) . . . . .	30
2.5.3	Hyperledger Fabric . . . . .	30

2.5.3.1	Membership . . . . .	31
2.5.3.2	Channels . . . . .	31
2.5.3.3	Execute-Order-Validate . . . . .	31
<b>2.6</b>	<b>Consensus protocols . . . . .</b>	<b>33</b>
2.6.1	Proof of Work (PoW) . . . . .	33
2.6.2	Proof of Stake (PoS) . . . . .	35
2.6.3	Other methods . . . . .	39
2.6.3.1	Proof of Activity (PoA) . . . . .	39
2.6.3.2	Ripple . . . . .	40
<b>2.7</b>	<b>Governance . . . . .</b>	<b>41</b>
2.7.1	Public vs Permissioned . . . . .	41
2.7.2	Governance models . . . . .	43
2.7.3	Forks . . . . .	45
<b>2.8</b>	<b>Blockchain security: attacks . . . . .</b>	<b>46</b>
2.8.1	Double spending . . . . .	46
2.8.2	Key security . . . . .	48
2.8.3	Sybil attacks . . . . .	49
2.8.4	The 51% attack . . . . .	50
2.8.5	Code vulnerabilities . . . . .	51
2.8.6	Denial of Service (DoS) . . . . .	52
2.8.7	Withholding attacks . . . . .	53
2.8.7.1	Withholding transactions . . . . .	53
2.8.7.2	Withholding blocks . . . . .	53
<b>2.9</b>	<b>Blockchain security: limitations . . . . .</b>	<b>54</b>
2.9.1	Bootstrap . . . . .	54
2.9.2	Privacy . . . . .	54
2.9.3	Propagation delay . . . . .	55
2.9.4	Transaction rate . . . . .	56
2.9.5	Legislation . . . . .	56
<b>2.10</b>	<b>Blockchain &amp; IoT . . . . .</b>	<b>58</b>
2.10.1	Use Cases . . . . .	59
2.10.2	Limitations . . . . .	61
<b>2.11</b>	<b>Conclusion . . . . .</b>	<b>61</b>

---

## 2.1 Introduction

The story of the blockchain starts with Bitcoin [Nakamoto, 2008], a cryptocurrency introduced in 2008 in a paper signed by a Satoshi Nakamoto. The first implementation was released a few month later in 2009. Bitcoin's goal is to replace trust, which is the foundation of a centralized system such as a bank, with a crypto-reliant system that makes fraud computationally hard. The attractiveness of this system resides in its distributed and public nature. In short, each node maintains a complete replica of a public ledger in which every transaction is written down. This eliminates the need for a third party and enables direct interactions between payers and payees, thus reducing the cost of a transaction. Anybody can join at any time.

Recent years have seen the blockchain leave the realm of cryptocurrency and take a life of its own. The blockchain technology is seen as really promising by many outside of the financial world. Ethereum kicked off the rise of the Blockchain 2.0 by introducing smart contracts and turning the blockchain into a *world computer* where any program can run, implementing a number of use cases. It is considered for numerous applications such as digital voting, supply chain management, grocery stores loyalty programs, digital identities, etc.

Concurrently, a new permissioned model has grown to satisfy the interest of the private sector in privacy and trade secrets. On permissioned blockchains, the right to perform operations can be restricted to a smaller number of entities. There is still a need for public blockchains however and both model co-exist happily in the ecosystem.

As a tool for decentralization, the blockchain shows promising applications in the IoT. It provides interesting properties such as desintermediation, transparency, and auditability. The blockchain can reduce deployment and maintenance cost, and provide a shared space for all kinds of applications IoT applications. This would enable the large IoT world to move as one. Access control in particular, could benefit from the flexibility and openness offered by the blockchain. It could be the tool to distribute authorization decision while sparing the low resources of edge devices.

The emergence of the blockchain also provides a new and exciting research area: the security and robustness of the protocol, consensus mechanisms, incentive systems, resource optimization, ... Many topics are just beginning to be explored and many more are yet to be discovered.

The goal of this chapter is to provide the reader with an introduction to blockchain concepts, security, and limitations. In particular, we discuss its legitimacy for IoT applications.

**Organization** Section 2.2 introduces the cryptographic primitives behind the blockchain. Section 2.3 presents the basic concepts one must get acquainted with in order to understand the blockchain. Section 2.4 proposes a formal definition of distributed ledgers. Section 2.5 discusses three of the main blockchain implementations, namely Bitcoin, Ethereum, and Hyperledger. Section 2.6 details several consensus protocols, their strengths and weaknesses. Section 2.7 addresses the complicated subject of governance in an open, public, and dynamic ecosystem. Section 2.8 presents blockchain attacks such as the famous double spending and 51% attacks. Section 2.9 discusses the current limitations of the technology from a security standpoint. Section 2.10 examines the pertinence of IoT blockchain applications. Finally, Section 2.11 concludes this chapter.

## 2.2 Cryptographic primitives

This section briefly presents the cryptographic primitives behind the blockchain. For a more in-depth presentation of hash functions and digital signatures, interested readers are referred to Chapter 9 and 11 of the *Handbook of applied cryptography* [Menezes et al., 1996], from which the following definitions are taken. Readers interested in Merkle trees are directed to Ralf Merkle’s seminal paper on the topic [Merkle, 1980].

### 2.2.1 Cryptographic hash functions

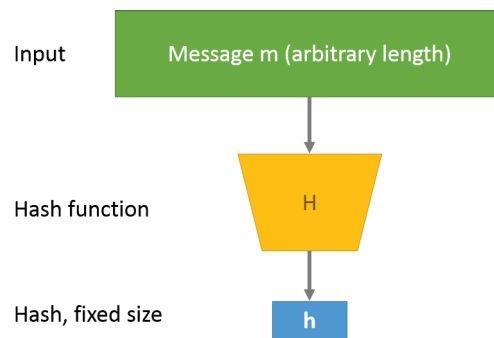


Figure 2.1: Hash function

Hash functions are widely used in computer systems.

**Definition 2.2.1** (Hash function). A hash function is a function  $h$  with the following properties

- **Compression** -  $h$  maps an input  $x$  of arbitrary finite length, to an output  $h(x)$  of fixed length  $n$ .
- **Ease of computation** - Given  $h$  and an input  $x$ ,  $h(x)$  is easy to compute

The output of such function is called a *hash*. Figure 2.1 illustrates the compression property of hash functions. In Definition 2.2.1, *easy* is purposefully left undefined as its meaning is context-dependent. It can be defined with regards to time, number of operations, or complexity for instance.

In practice, hash functions also require the following property: a difference in input, however small, should lead to noticeably different outputs. Figure 2.2 uses MD5, a common hash function, to showcase the impact that a one-letter difference (between the first and third inputs) has on a message’s hash. This enhances usability by allowing humans to easily spot the hash difference.

So a hash function yields an easy to compute, short, fixed-length representation of the input data. These hashes can be used to compare files, find duplicate records, or accelerate data lookups. They are also used in cryptography.

Cryptographic hash functions are at the heart of modern cryptography. They are used to guarantee data integrity and message authentication. As such, they require stronger properties than simple hash functions.

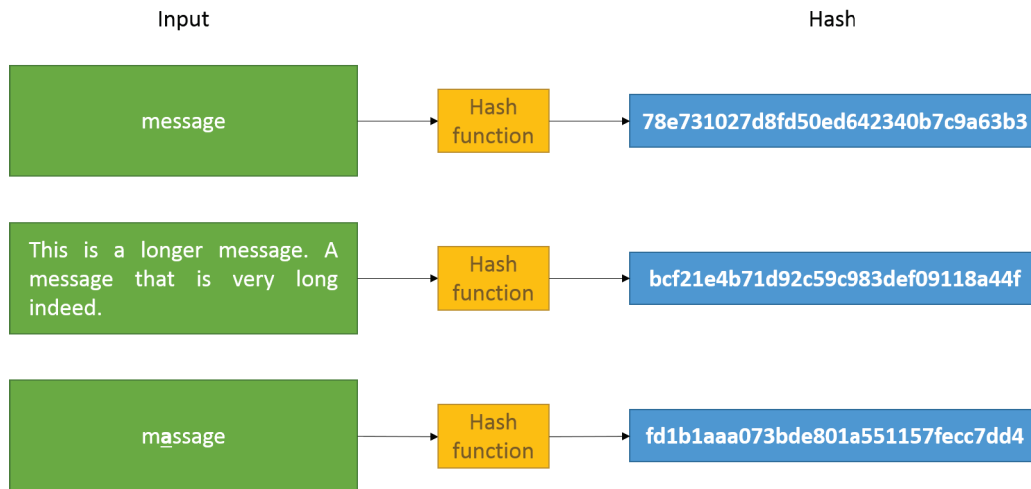


Figure 2.2: Applying MD5 to different inputs

**Property 2.2.1.** Let  $h$  be a hash function. We define the following properties:

1. **Preimage resistance** - For any pre-specified output  $y$  it is computationally unfeasible to find an input  $x$  such that  $h(x) = y$ .
2. **2nd-preimage resistance** - Given an input  $x$ , it is computationally unfeasible to find any  $x' \neq x$  such that  $h(x) = h(x')$ .
3. **Collusion resistance** - It is computationally unfeasible to find any two inputs  $x \neq x'$  such that  $h(x) = h(x')$ .

Here again, the term *computationally unfeasible* is context-dependent. Hash functions that are both preimage and 2nd-preimage resistant are called *one-way hash functions* or *weak hash functions*. Hash functions that are both 2nd-preimage and collision resistant are called *collision resistant functions* or *strong hash functions*. Note that collision resistance does not imply preimage resistance. The identity function for instance, is both 2nd-preimage resistant and collision resistant but finding a preimage is trivial. However, in practice, collision resistant hash functions are constructed to be preimage resistant as well.

One-way hash functions and collision resistant hash functions are cryptographic hash functions. In the remainder of this thesis, we will generally use cryptographic hash functions to refer to collision resistant hash functions.

These properties guarantee that, given a hash, an adversary cannot produce an input that is different from the original message while still hashing to the same output. It is also really hard to find two messages that yield the same hash. A hash can therefore be considered as a **unique** representation of the input message, giving us a short, fixed-length pointer to data of any size.

This representation, or digest, can then be used for digital signature.

### 2.2.2 Digital Signatures

Digital signatures are the equivalent of the handwritten signature one might find at the bottom of a document. Like their paper-based counterparts, digital signatures must be:

- **Unforgeable** - No one must be able to produce a valid signature for an entity  $S$  but  $S$  themselves,
- **Verifiable** - Anyone with the correct information must be able to associate a signature with its emitter,
- **Non-repudiable** - A signer  $S$  cannot successfully dispute the origin of their signature.

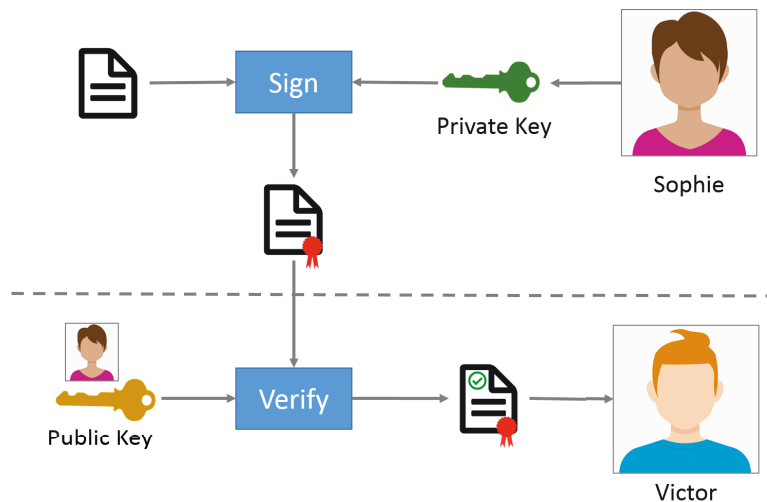


Figure 2.3: Digital Signature

A digital signature scheme is composed of three algorithms:

- **Key generation algorithm** - A method for generating a public/private key pair used for signing.
- **Signing algorithm** - A method for producing a digital signature.
- **Verification algorithm** - A method for verifying that a digital signature is authentic (i.e., was indeed created by the specified entity).

To be used in practice, a signature should be easy to compute by the signer, easy to verify by anyone, and have a security level appropriate to the expected time relevance of the signature: when signing a testament, the signature needs a high level of security as the authenticity of the document will be relevant in a matter of years (hopefully), when signing a message fixing a meeting in a weeks time, the signature loses relevance once the meeting has passed or even when the meeting has been arranged.



**Key generation** A digital signature scheme requires two keys. The first is private and used to sign the message. The second is public and used to verify the signature. In order to produce a verifiable signature, a signer  $S$  must therefore first generate a key pair  $(priv_S, pub_S)$  and make the public key available to those who will later verify their signature.

Anyone that comes into possession of  $S$ 's private key can sign in their name. The security of the scheme then rests on the security of the private key.

**Signature** As illustrated in Figure 2.3, a signing algorithm takes two inputs, the message and the private key, and outputs a signature. An example of signature scheme is to encrypt the cryptographic hash of the message with the private key. Anyone with the public key can decrypt the signature, recompute the message's hash, and compare the two.

**Verification** As illustrated in Figure 2.3, the verification algorithm takes the public key and the signature as inputs and outputs a boolean. Some schemes enable the retrieval of the original message from the signature. When it is not the case, the verification algorithm requires the original message as an additional input.

The verification process confirms that the message has indeed been signed by the private key associated to the inputted public key. Ownership of these keys however must be proven through other means.

### 2.2.3 Merkle trees

In a Merkle tree [Merkle, 1980], also called hash tree, each leaf is labelled with the cryptographic hash of a data block. The tree is then constructed by labelling each non-leaf node with the cryptographic hash of its child nodes. A tree is usually smaller than the data it represents thanks to the *compression* property of hash functions. Hash trees also encode data in a privacy-preserving fashion as no information from the original data can be recovered from the tree thanks to the *preimage resistance* of cryptographic hash functions.

Figure 2.4 illustrates the encoding of data into a Merkle tree. The original data blocs (in lighter blue) are not part of the tree. A data block can represent part of a larger file, or be an element in a set. Both cases are explored in the following use cases.

**Use Case 1 - Peer to peer file sharing** Merkle trees are used to verify the integrity or authenticity of documents. Let us take the example of Alexander that is trying to retrieve a file from a peer to peer system. In this system, a hash tree is computed for the file by breaking it down into data blocks, that are then hashed to form the leafs. These data blocks also correspond to the blocks that will be received from peers when downloading the file. The root of the tree is then added to the file's metadata.

Before requesting the file, Alexander retrieves the corresponding Merkle tree root from a trusted peer, Belinda. He then retrieves the file broken down into data blocks from untrusted peers. Upon reception, Alexander computes the Merkle tree of the file and compares its root to the hash he obtained from Belinda.

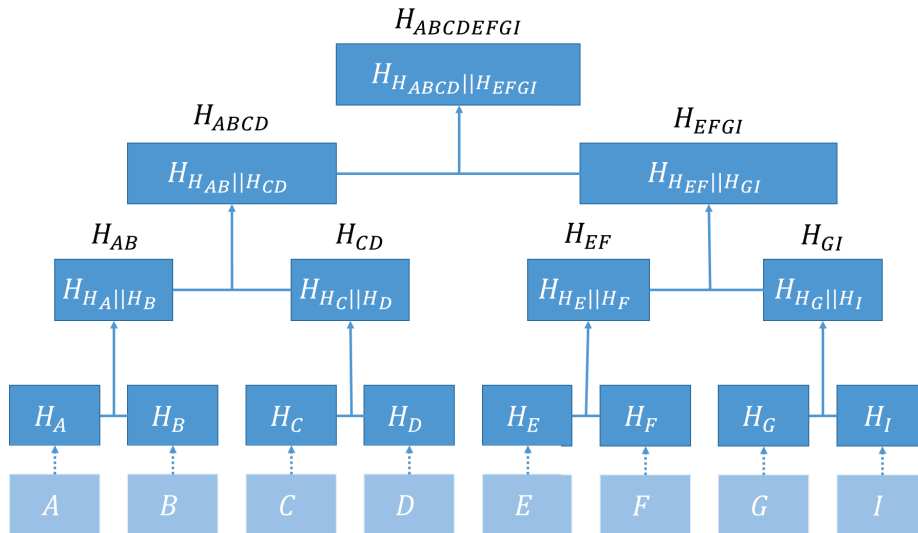


Figure 2.4: Merkle tree

Figure 2.5 shows how any modification of an original data block impacts the root of the tree, as well as intermediary nodes. If the roots do not match, Alexander concludes that the file he downloaded is not the one he wanted. This may be due to malice, or to errors during transfer. In any case, Alexander can contact another peer and try again.

Merkle trees enable the verification of partial data: by retrieving intermediary nodes of the Merkle tree in addition to the root, Alexander can verify data blocks are legitimate without downloading the whole file. This is illustrated by the next example.

**Use case 2 - Election verification** A Merkle tree can also be used to provide a proof of membership. Let us consider an election. A voter should be able to verify that their vote has been tallied up. But no one should have access to another person’s vote. Figure 2.6 shows how encoding votes in a Merkle tree answers both requirements.

When the election is over, each vote is added to the tree and the root is publicly posted as a representation of the election. Now Daniel wants to verify that their vote is included in the tree. The entity in charge of the election isolates the path from  $D$  to the root and sends the corresponding intermediary nodes to Daniel (in green in Figure 2.6). From his vote, Daniel computes  $H_D$ . Using the provided  $H_C$ , Daniel computes  $H_{CD}$ , and so on until he reaches the root that can now be compared to the value that has been publicly advertised.

To successfully cheat, an adversary would need to produce a fake path in the tree that would hash to the same root. The preimage resistance of cryptographic hash functions ensures that the adversary cannot compute a valid preimage from the root. Collusion resistance ensures that the tree does not contain a data block  $X$  such that  $H_X = H_D$ . The adversary therefore cannot create such a path.

Data blocks that are no longer relevant can also be removed from the tree without affecting the

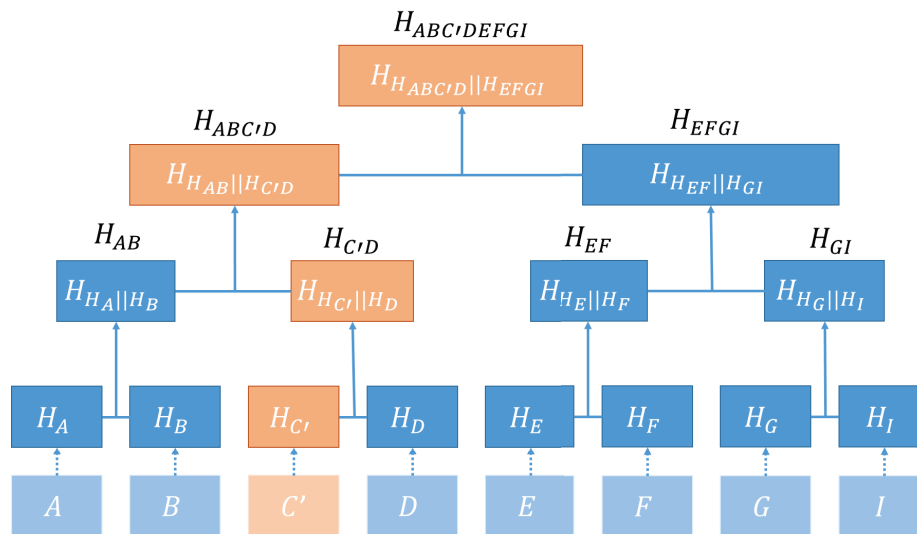


Figure 2.5: Merkle tree: modifying a data block

membership verification process if intermediary hashes are kept. This is used in the blockchain to limit the size of the blockchain history.

## 2.3 Basic Blockchain Concepts

A blockchain is a shared ledger compiling transactions (tx). *Blockchain* is the name given to both the peer-to-peer network and the records it creates. Figure 2.7 provides an overview of the steps taken to integrate a transaction into a blockchain. The steps are as follows:

- **Panel 1:** A new transaction is created and broadcasted to the network.
- **Panel 2:** Miner nodes (a.k.a miners) verify the transaction. If valid, miners collect the transaction and add it to their pool of valid but yet-to-be-approved transactions.
- **Panel 3:** Miners work on creating a block (see Section 2.6 for consensus protocols) including the transactions in their transaction pool.
- **Panel 4:** A new block is proposed, verified, and propagated in the network.
- Miners start collecting transactions and working on the next block.

The present section introduces basic terms and concepts necessary to understand the blockchain. They will be used throughout this thesis.

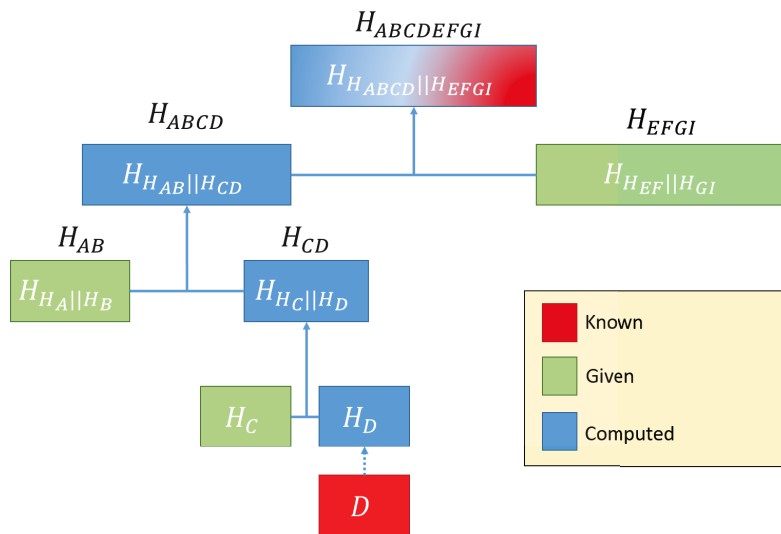


Figure 2.6: Merkle tree: proof of membership

### 2.3.1 Transactions

In essence, a blockchain is a transaction ordering mechanism. These transactions can describe a transfer of assets (cryptocurrency or other), or an interaction with a smart contract. Transactions must always be signed by their emitter. They can be prepared offline and are then broadcasted to the peer-to-peer network that composes the blockchain. Miners will verify them, before bundling them into a block. A transaction is refused if its digital signature is invalid, if it conflicts with the blockchain history, or if it otherwise breaks blockchain rules. Their specific format is implementation-dependent.

### 2.3.2 Addresses

In order to send or be the recipient of transactions, a user must have a blockchain address. To create an address, a public/private key pair is generated for the user. The public key is hashed to create the user's blockchain address. The private key is used to sign outgoing transactions. One can create as many addresses as they want. Each new address acts as a new pseudonym. This increases the user's privacy by making it more difficult for people to trace their activity.

Smart contracts have addresses that are used to invoke their functions.

### 2.3.3 Miners

Some nodes in the network dedicate resources to verifying transactions and maintaining the security of the blockchain. They are called miners. Miners are paid by block rewards. For each new block, a block reward is awarded to a single miner or a small group of them. This reward system is the only way new

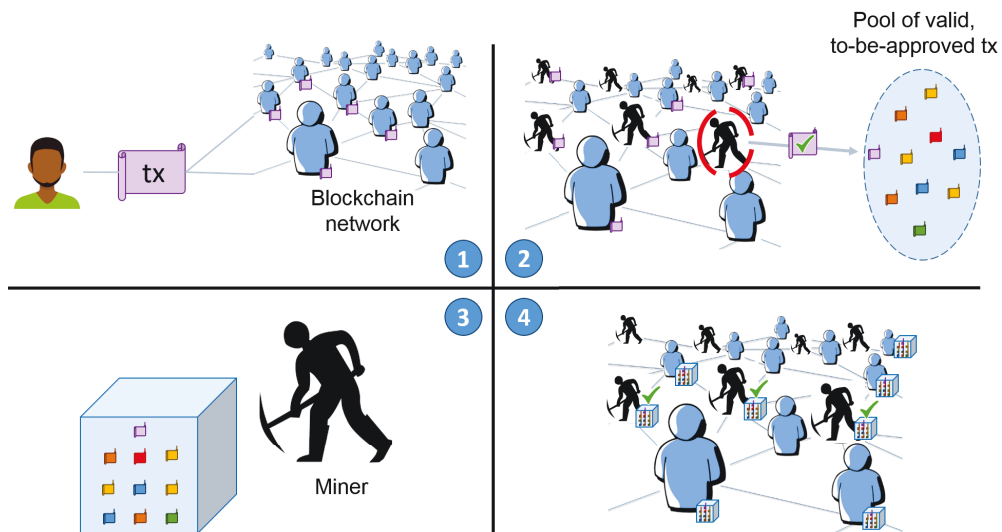


Figure 2.7: Overview of the blockchain

coins can be created in the system. Transactions can (and usually do) include a transaction fee. The sum of all of the block's transaction fees included in a block are added to that block's reward.

The specifics of block rewards are implementation-dependent. In Bitcoin for instance, the block reward was originally of 50 BTC per block and halves every 210 000 blocks, which take roughly 4 years to mine. At the moment, it is therefore down to 12.5 BTC per block. Transaction fees supplement this lost in income. In Ethereum, miners also have to run smart contracts. Users must include a fee proportional to the difficulty of the operation. Block rewards are only a few ETH, but blocks are produced every few seconds rather than every 10 minutes.

Miners compete to receive the reward by participating in the consensus protocol. The more miners participate, the more overall resources are poured into the consensus, the harder it is for an individual to single-handedly match that and gain too much influence over the blockchain.

### 2.3.4 Blocks

A block is an assembly of ordered transactions. It is composed of a header, and a body. The body contains the transactions and a Merkle tree formed by their hashes. The header contains the block's meta data, including the root of the Merkle tree.

Blocks are linked together to form a chain. The first block is called the *genesis* block or block 0. Blocks are identified either by the hash of their header, or by their height, i.e. the distance between them and the genesis block. The *chain* is created by including the hash of block  $n - 1$  in the header of block  $n$ . So in Figure 2.8, block 11's header contains the hash of block 10 while block 12 contains the hash of block 11.

Any modification of the content of a block modifies its hash. If an attacker modifies the block  $i$ , its

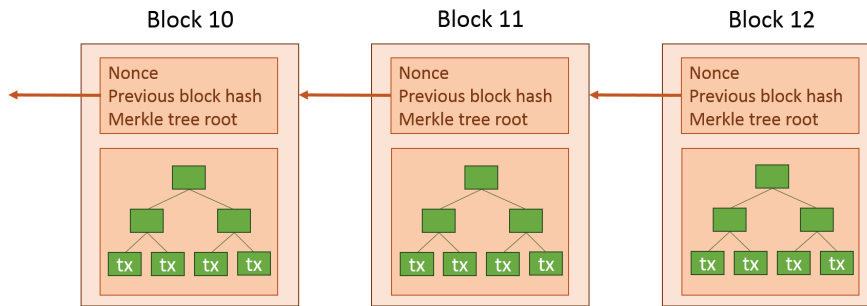


Figure 2.8: Blocks

hash no longer equates the value stored within block  $i + 1$ . One can therefore notice the modification occurred by comparing the two values. To hide this, the attacker needs to also modify block  $i + 1$ , which would force them to modify block  $i + 2$  to replace the now modified hash, and so on, and so forth, until they reach the most recent block. Blockchains are set up so that producing a valid block is hard. For each block stacked on top, the task becomes even harder for an attacker. This is the heart of the tamper-proof nature of the blockchain, along with its distributed architecture.

A transaction is considered as verified when it has been included in a block. Because the last few blocks are subject to modification (see Section 2.8.1), users should wait until a transaction is buried under several blocks before considering it as definitively integrated into the blockchain. Each new block added to the chain on top of a given block diminishes the chances that this block will be removed.

### 2.3.5 Smart Contracts

The term smart contract was coined by Nick Szabo in the 1990s. In a paper from 1997 [Szabo, 1997], he takes the vending machine as the example of an archaic smart contract : The machine enforces a contract between the buyer and the vendor and makes the breach of this contract prohibitively expensive in terms of engaged efforts, as breaking the machine is far more complicated than simply paying a few euros. This is the essence of smart contracts. They are programs or protocols that facilitate the enforcement of a contract. This contract might have legal value but can also be taken in the simpler sense of an agreement. Ethereum [Buterin et al., 2013] was the first blockchain to implement smart contracts.

One key aspect of blockchain smart contracts is their immutability. Once deployed, their code is written down in the common ledger. As such, they cannot be unilaterally modified unless they were designed to authorize updates. Everyone with a copy of the blockchain can therefore consult the smart contract's code and verify its execution.

A user can interact with a smart contract in two ways : by making a call or by sending a transaction. A call does not modify the blockchain state. It is a read operation and can be executed offline. If the user wishes to modify the blockchain state, a blockchain transaction must be sent out to the network. As with every transaction, miners will verify it by running the corresponding code and, if successful, update the blockchain state.

To illustrate this separation, we take the example of an election implemented using a smart contract. It presents two functions: *vote* and *seeResult*. Voting will modify the state of the blockchain. A transaction must be sent out and verified by the miners before a vote can be taken into account. Viewing the result of the election however, does not. A simple call to one's local version of the ledger will do.

Miners use computational power to run transactions on smart contracts. A fee is therefore required that is proportional to the work they entail. If the fee contained within the transaction is not enough to complete the execution, the execution stops but the money is not returned. This acts as a protection against Denial of Service (DoS) attacks where an attacker would, for instance, try to run an infinite loop, thus mobilizing all of the miners resources. Storage of data into a smart contract must also be paid for in proportion with the requested memory space. This cost is paid by the user that sends the transaction and not by the user that deployed the smart contract in the first place.

### 2.3.6 Forks & Longest Chains

Sometimes, two miners will find a block at roughly the same time. These two valid blocks have the same parent block and therefore correspond to the same place in the chain. There is often no logical reason to prefer one over the other. The network is therefore presented with two alternate but equally valid versions of history. This is called a fork.

This is where the notion of longest chain comes into play. Rather than a chain, a blockchain is effectively a tree with the genesis block as its root. Miners only work on the longest path (or longest chain) from root to leaf and it is the only valid version of the blockchain history. The definition of *longest* chain varies with the blockchain and the consensus protocol it uses. It can simply be the path with the most blocks, or, for Proof of Work (see Section 2.6.1), additionally take the puzzle difficulty of each block into account.

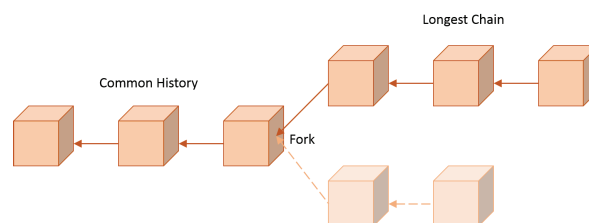


Figure 2.9: Blockchain fork

Forks are solved over time. When the next block is created, the block it chooses as its ancestor becomes part of the longest chain and the concurrent blocks are dropped. This is illustrated in Figure 2.9. Forks occurrence rate and resolution time depend on the block rate, network size (see Section 2.9.3), and consensus protocol.

## 2.4 Formal definition

For the scientific community, the blockchain is some kind of oddity. Nakamoto did not give a formal definition in the original paper and many argue that it shouldn't work. And yet it does. This absence of a theoretical model makes for a complicated analysis of the system, including its security. Pass et al [Pass et al., 2017] propose an abstract definition of the blockchain and show that Nakamoto's blockchain fits this definition.

Authors define the blockchain as an interactive protocol where each participant receives inputs they try to include in their local chain. The following properties are required :

**Property 2.4.1.** *Let  $T$  be a positive natural number.*

1. **Consistency** - *With overwhelming probability (in  $T$ ), at any point, the chains of two honest participants can only differ in the last  $T$  blocks.*
2. **Future self-consistency** - *With overwhelming probability (in  $T$ ), at any two points in time  $r, s$ , the chains of any honest participant at  $r$  and  $s$  differ only in blocks within the last  $T$  blocks.*
3.  **$g$ -chain-growth** - *With overwhelming probability (in  $T$ ), at any point in the execution, the chain of honest participants grew by at least  $T$  messages in the last  $\frac{T}{g}$  rounds, where  $g$  is called the chain-growth of the protocol.*
4.  **$\mu$ -chain-quality** - *With overwhelming probability (in  $T$ ), for any  $T$  consecutive messages in any chain held by some honest participant, the fraction of messages that were contributed by honest participants is at least  $\mu$ .*

The first property, consistency, is somewhat obvious. It ensures that, natural forks aside, the network is in agreement and was formalized by Garay et al [Garay et al., 2015]. The second property, future self-consistency, ensures that the chain cannot be drastically modified overnight. The counter-example taken by Pass et al [Pass et al., 2017] is an application that keeps switching between two chains. Future self-consistency forbids such behaviour and ensures consistency over time. The third property,  $g$ -chain-growth, ensures that the chain keeps growing at a steady rate, allowing new transactions to be included. It was considered in both Sompolinsky and Zohar [Sompolinsky and Zohar, 2015] and Garay et al. [Garay et al., 2015]. Finally, the last property ensures that honest participants still control a known fraction of the process. It was first discussed on the Bitcoin forum and later formalized by Garay et al. [Garay et al., 2015].

Pass et al. [Pass et al., 2017] show that verifying these properties is enough to implement a public ledger.

## 2.5 Blockchain Implementations

The blockchain exists in many forms. In this section, we present three of the most famous blockchain implementations: Bitcoin, Ethereum, and Hyperledger Fabric. We delve into the details of their inner working to see how they differ and what each of them brings to the table.



## 2.5.1 Bitcoin

Launched in 2009 by the mysterious Satoshi Nakamoto, Bitcoin remains the most popular blockchain to date. Bitcoin is both the name of the protocol and that of the currency, which symbol is BTC. The seminal paper [Nakamoto, 2008] explaining its general concepts was originally published on a cryptography mailing list. Bitcoin benefits from an active community of developers and contributors that helped refined the concepts even in the early days.

### 2.5.1.1 Transaction-based model

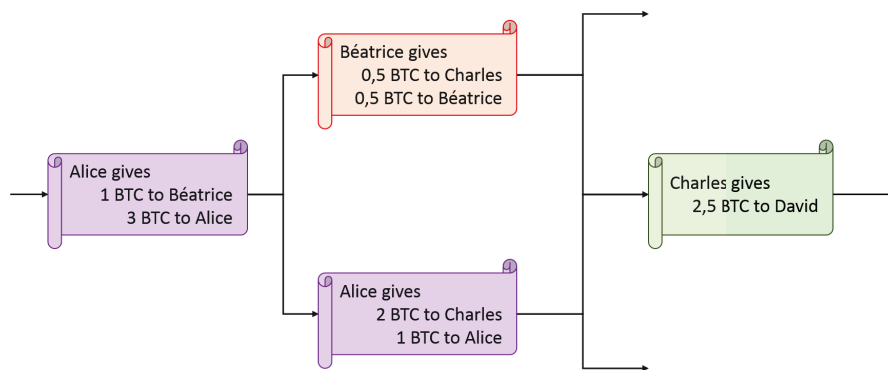


Figure 2.10: Bitcoin transaction model

Bitcoin's goal is to replicate cash. As such, no account balance is kept. Instead, transactions are linked to each other as the output of one becomes the input of another and so on. The same way a bill exchanges hands and gets split into smaller denominations as it makes its way through the economy. So the question is never "How much do you have in your bank account?" but rather "Can you produce enough bills to pay for this?".

Let us consider the following case. Alice has 4 BTC. Figure 2.10 illustrates how these coins can be exchanged. Alice needs to pay 1 BTC to Béatrice. She does so by referencing the transaction that originally gave her the 4 BTC (as an input), and gives 1 of them to Béatrice, the rest to herself as change. That transaction has one input and two outputs. A transaction must always consume the entirety of the original balance. If not null, the difference between input and output is given away to miners as compensation.

BTC can be broken down as illustrated by Béatrice's payment to Charles (in red in Figure 2.10). The smallest accepted denomination is the satoshi, which corresponds to  $10^{-8}$  BTC. A transaction can also accept any number of inputs, as illustrated by Charles transaction (in green in Figure 2.10). It takes two inputs (Béatrice's and Alice's transaction outputs) and generates one output to David. Inputs can come from different addresses, as long as each is signed by the corresponding key. There is no limit on the number of outputs nor recipients.

To spend bitcoins, one must be able to produce Unspent Transaction Outputs (UTXO) addressed to

them. Once an output has been named as an input in a transaction, that output is considered spent. Any transaction that references it will be deemed invalid. The bill has changed hands.

Note that no one ever actually holds anything. Bitcoins only exists as the result of transactions in the ledger. Users are merely pointing at the shared ledger, saying *"See, that guy gave 2 BTC to me. I'm giving them to this other guy. Mark it down !"*.

### 2.5.1.2 Wallets

Wallets hold the public/private key pair necessary to spend UTXO. They are not part of the core protocol and theoretically optional. Many different options exist on different platforms or devices. They can be physical devices holding the user's private key in a smartcard, mobile applications that require a QR code to be scanned, a computer software with a password, a service provided by a trusted third party, etc.

They can be used to generate multiple addresses from a single master key, thus increasing the user's privacy by providing several pseudonyms for them. Wallets are also a simpler way to keep track of one's UTXO.

Weaknesses in wallet software have been used to steal cryptocurrencies in the past.

### 2.5.1.3 Scripts

UTXO are locked by scripts. They describe the conditions one must fulfill to spend the corresponding output. When creating a transaction, the payer specifies a script for each output. This script contains, for instance, the blockchain address of the recipient. When referencing a UTXO in a new transaction, the user must provide the correct arguments for the script to terminate successfully.

The most common example requires the user to prove they own the private key associated to a specific Bitcoin address. For this, the user provides:

1. A public key, that hashes to the specified Bitcoin address,
2. A valid signature to prove ownership of the associated private key.

Bitcoin scripting language is stack-based, and is intentionally not Turing complete. It provides flexibility. A script can, for instance, require multiple signatures to be unlocked, freeze funds for a specific time, make the money unspendable, require the spender to solve a puzzle, etc.

### 2.5.1.4 Alt-coins

Altcoin (for alternative coin) is the name given to any cryptocurrency that is not Bitcoin. There are a number of them. Some are included in Figure 2.11. Bitcoin is open source. Anyone can therefore create their own cryptocurrency based on Bitcoin's code, with any alteration they would like.

Dogecoin for instance, was introduced in 2013 as a joke. It takes its name from an internet meme featuring a breed of Japanese dog renamed *Doge* by the Internet. A key difference with Bitcoin would be the total number of coin in circulation: Bitcoin will cap at 21 millions coins, Dogecoin over 100 billions.



Figure 2.11: Logos of various cryptocurrencies, including Doge coin

## 2.5.2 Ethereum

Launched in 2015, Ethereum is the first blockchain to introduce smart contracts. Its associated cryptocurrency is the Ether (ETH). The workings of Ethereum are exposed in its white and yellow papers [Buterin et al., 2013, Wood, 2014], written respectively by Vitalik Buterin and Gavin Wood, its co-creators. It presents itself as a blockchain with a Turing-complete programming language, *Solidity*. The blockchain is seen as a state transition system where each valid transaction represents a state transition function. This departs from Bitcoin model where no state is kept.

In Ethereum, the block time averages 15 seconds, though it uses the same kind of consensus as Bitcoin (namely Proof of Work). The block reward is constant. Ethereum's development was crowd funded. This resulted in 72 millions ETH coins being distributed to early investors when the blockchain launched. In 2019, this still accounts for more than half of the total circulating supply.

### 2.5.2.1 Accounts

Two kinds of accounts exist : externally owned accounts and contract accounts. The latter are held by smart contracts and controlled by their code while the former corresponds to blockchain users and are controlled by private keys. Both are associated with an ether balance. The account corresponding to a contract also holds the code of this contract and a storage. An account is created if it is the recipient of a transaction and does not yet exist.

### 2.5.2.2 Messages and Transactions

In Ethereum, we differentiate between transactions and messages. Transactions are sent from externally owned accounts and lead to the sending of a message. They are signed with the sender's private key. Transactions can be used to send funds, deploy smart contracts, or execute their functions.

When a contract account receives a message, its code is executed accordingly. It might lead to the contract sending a message of its own to another contract. A message contains an optional data field

that will be interpreted by the recipient according to its specification. This field is used, for instance, to provide arguments to a smart contract's function.

Another difference between messages and transactions is that messages will not be stored in the blockchain, only transactions will. Messages only exist during execution. This means that when a transaction triggers a chain of messages between smart contracts, it might be hard to trace the incurring modifications back to the original transaction.

### 2.5.2.3 Crypto-fuel

The ether (ETH) is designed to be used as a crypto-fuel, or gas. This fuel pays for the execution of code. Which is why a message contains a *STARTGAS* value corresponding to the maximum amount of gas available for any computation that may occur as a result. Each atomic instruction such as a comparison, or storing a value, costs a fixed amount of gas. This is to prevent infinite execution. Indeed, running an infinite loop would require an infinite amount of gas. All execution is therefore bound to halt.

Data sent along with a message also cost gas. When a contract runs out of gas and the code is not done executing, all changes are reverted and the engaged ether is lost. If the execution consumed less gas than what was provided, the excess is returned to the sender.

Gas prices can vary. Transactions include a *GASPRICE* field for the sender to indicate the price they are willing to pay by computational step. The idea is to force an eventual attacker to pay proportionally to the resources it uses. It also allows the network to react to attacks by momentarily raising the price of gas, thus influencing the cost of said attack. Finally, gas price can adapt to the fluctuations of the ETH value, providing miners with a consistent rate. If the sender doesn't have the necessary funds in its accounts, the call will fail.

The fees that are paid for gas will be added to the block reward.

### 2.5.2.4 The Ethereum Virtual Machine (EVM)

Ethereum contracts are written in a high level language such as Solidity and are then compiled into bytecode before being deployed in the blockchain through a transaction. The Ethereum Virtual Machine is in charge of running that code. It is stack-based. During execution, the code is stored either on the stack, in memory or in the contract account storage. The first two will be whipped clean after the process ends.

Code execution is part of transaction validation. The code is therefore executed by **all** nodes that validate transactions. The state of the system, which contains all accounts, is stored in the blocks.

## 2.5.3 Hyperledger Fabric

The Hyperledger project is an open source initiative by the Linux Foundation started in 2015 to enable cross-industry collaboration in developing blockchain applications and their surrounding tools. The goal is to address the scalability and reliability limitations of existing blockchain solutions to facilitate their adoption for industrial applications.

Hyperledger Fabric [Androulaki et al., 2018] is one of many projects under the Hyperledger umbrella. It was originally proposed by IBM in 2016. Where Bitcoin and Ethereum are both public blockchains,

Hyperledger Fabric enables restrictions on operations such as reading the blockchain state or sending a transaction. It supports smart contracts (called chaincodes), several consensus protocols, and membership services.

Fabric is built for modularity, to better adapt to the needs of various industrial use cases. This translates to switchable consensus protocol and a flexibility in chaincode development. Contrary to Ethereum that uses its own language for smart contracts, Fabric supports general-purpose programming languages for its chaincodes. Furthermore, Fabric does not have a backing cryptocurrency. This is mostly due to its permissioned model (see Section 2.7).

### 2.5.3.1 Membership

Nodes must enroll to use Fabric. Enrollment is handled by the *Membership Service*. A node can be one of three things : a client, a peer, or an orderer. Some peers can also act as endorsing peers (or endorsers) depending on the transaction.

*Clients* act on behalf of users and submit transaction proposals. Transactions revolve around deploying or calling chaincodes. *Peers* execute and validate these proposals. For each transaction, the associated chaincode determines which subset of peers can validate the proposal. Those who do are called *endorsers*. *Orderers* collectively form the ordering service. They are in charge of running the consensus protocol that establishes the total order of transactions.

The Membership Service is built to support different implementations. At the moment, the default is a Public Key Infrastructure (PKI). Fabric offers support for federation where several organizations use the same network and operate their own Membership Service. Credentials can be distributed online or offline.

### 2.5.3.2 Channels

Fabric is divided into channels. One channel corresponds to a ledger. A node can have access to any number of channels and maintain a separate ledger for each.

Let us take the example of an apple company selling to different grocery stores. The company sets different prices for each store, based on the volume purchased, the history between the two companies, or any other reason. Using a public blockchain, each store can see how much the others are paying, leading them to renegotiate their prices. This is not to the apple company's advantage. Using channels however, each price can be set on a private channel. The rest of the operation such as shipping, deliveries, and so on can be on a larger channel containing every store.

This information separation enhances user privacy and opens the door for industrial applications with privacy requirements.

### 2.5.3.3 Execute-Order-Validate

Another difference between Fabric and the other two is the order in which the flow of transaction validation operations occurs. In Bitcoin and Ethereum, transactions are pre-executed by miners, compiled into a block, and propagated throughout the network to be executed again by all peers *in order*. For a given miner that did not produce the block, the order may differ from its own. Transaction *A* therefore

can yield a different result when applied after being enclosed in a block from when it was first received by the miner and pre-executed to check its validity. The transaction remains valid, but the resulting state might differ. This is defined as the *Order-Execute* Architecture.

In Fabric, the steps are reversed. Transaction validation can be broken down into three phases : Execution, Ordering, Validation.

**Execution Phase** A client that wants to invoke a chaincode makes a transaction proposal. The proposal is then submitted to that chaincode's endorsers. Endorsing requirements vary between chaincodes. All endorsers may be required to endorse, or only a portion of them (5 out of 15 for instance), etc. There is a great deal of liberty here.

Let us consider the following scheme: each endorser is given a share such that the sum of all shares is 100. A transaction proposal is valid if it is endorsed by at least 50% of the shares. Now  $a$  has 20% of the shares,  $b$  and  $c$  each have 10,  $d$  has 25,  $e$  and  $f$  have 15 each, finally  $g$  has 5.  $(a, d, g)$ ,  $(b, c, e, f)$ , and  $(a, b, c, f, g)$  are all valid endorsement sets.

When contacted, an endorser checks the proposal's signature. It simulates its execution and endorses the proposal by apposing its signature. The value of the input at the time of execution (as read from the channel) as well as the channel state after the execution are part of the returned endorsement.

The strategy of which endorsers to contact and in what order is left to the client's appreciation. Once enough endorsements have been gathered, the client forms a transaction assembling its proposal and the endorsements, and submits it to the ordering service.

**Ordering Phase** The Ordering Service does not verify the validity of the transaction it receives. It simply provides a total order for transactions on a per channel basis. Transactions are grouped by blocks and broadcasted to nodes of their respective channels. A block is identified by its sequence number. It contains a list of transactions and a *hash chain* value  $h$  that corresponds to the cryptographic hash of the previous block. Nodes can query the Ordering Service to retrieve past blocks.

The Ordering Service guarantees the following properties [Androulaki et al., 2018]:

- **Agreement** - Two blocks delivered by correct peers with the same sequence numbers are equal.
- **Hash chain integrity** - If the blocks  $B_s$  and  $B_{s+1}$  are both delivered by correct peers then it holds that  $h' = H(B_s)$ , where  $h'$  is the hash chain of  $B_{s+1}$  and  $H(\cdot)$  is the cryptographic hash function.
- **No skipping** - If a correct peer delivers a block with number  $s > 0$ , then this peer has already delivered a block for all  $i$  such that  $0 \leq i < s$ .
- **No creation** - When a correct peer delivers a block, all transactions contained within have been submitted by some client.

Additionally, for liveness-sake, the following property is desirable :

- **Validity** - If a correct client submits a transaction to the Ordering Service, then every correct peer will eventually deliver a block that contains it.

A *No Duplication* property is also desirable though optional. The Ordering Service may perform access control checks to verify whether a client is allowed to submit and broadcast a transaction, or receive blocks from a given channel.

Any consensus protocol that verifies these properties can be implemented in Hyperledger Fabric. The ordering method can even be switched over the life of the blockchain. It does not maintain the blockchain state nor does it control the validity of the ordered transactions.

**Validation Phase** After being ordered, transactions are broadcasted to their corresponding channels where they are validated by peers. The validation process consists in three steps :

1. The endorsements are checked against each chaincode's endorsement policy.
2. Endorsed inputs are checked against current values as read from the ledger.
3. The ledger is updated.

If step 1 or 2 fails, the transaction is marked as invalid. It is still included in the ledger but does not modify the channel's state.

## 2.6 Consensus protocols

One of the most important challenges of the blockchain is to keep the different copies of the ledger consistent with one another. This is achieved through a consensus algorithm.

There exist two main types of such protocols currently in use: Proof of Work (PoW) and Proof of Stake (PoS). All aim at electing a leader that will propose the next block to the network. Other propositions have some merits such as hybridizing PoW and PoS to mitigate the shortcomings of both, or basing the protocols on social interactions. All of these protocols are explained below. The quest for the optimal consensus process is still underway [[Vukolić, 2015](#)].

### 2.6.1 Proof of Work (PoW)

Proof of Work is the consensus mechanism used in Bitcoin and was summarized in the original paper as "one CPU, one vote". It is the most mature solution to date and is based on hash functions. PoW is hard to produce, but easy to verify.

**How it works** Miners try to solve a computational puzzle: In addition to the root of the transaction tree and the hash of the previous block, a block's header contains a nonce. The nonce value is chosen such that the block's header hashes to a value smaller than a specified target. The first miner to find a valid nonce propagates their now valid block throughout the network. Upon reception, nodes verify the block by hashing it, which is easy by [Definition 2.2.1](#). They then verify all transactions contained within. If all checks pass, miners advertise the new block to their neighbours and the propagation continues until all the nodes have received it. This block becomes the new head of the blockchain and miners start working on the next one.

Based on the preimage resistance of hash functions (see Property 2.2.1), the PoW puzzle can only be solved by brute force, i.e. trying nonces until one works. The more computational power an entity dedicates to solving this puzzle, the greater its chances. The first to find a solution collects the reward. The difficulty of the puzzle adapts to the number of participants: if the puzzle is becoming too easy, its difficulty will increase. When fewer miners participate and the puzzle is too hard, the difficulty will lower. In Bitcoin, this re-evaluation occurs every 2016 blocks, which corresponds to roughly 2 weeks.

The more miners participate in the PoW, the more computational power is invested and the more difficult the puzzle. An attacker would need more computational power of its own to compete against honest miners. A block containing invalid transactions will be rejected even with a valid nonce.

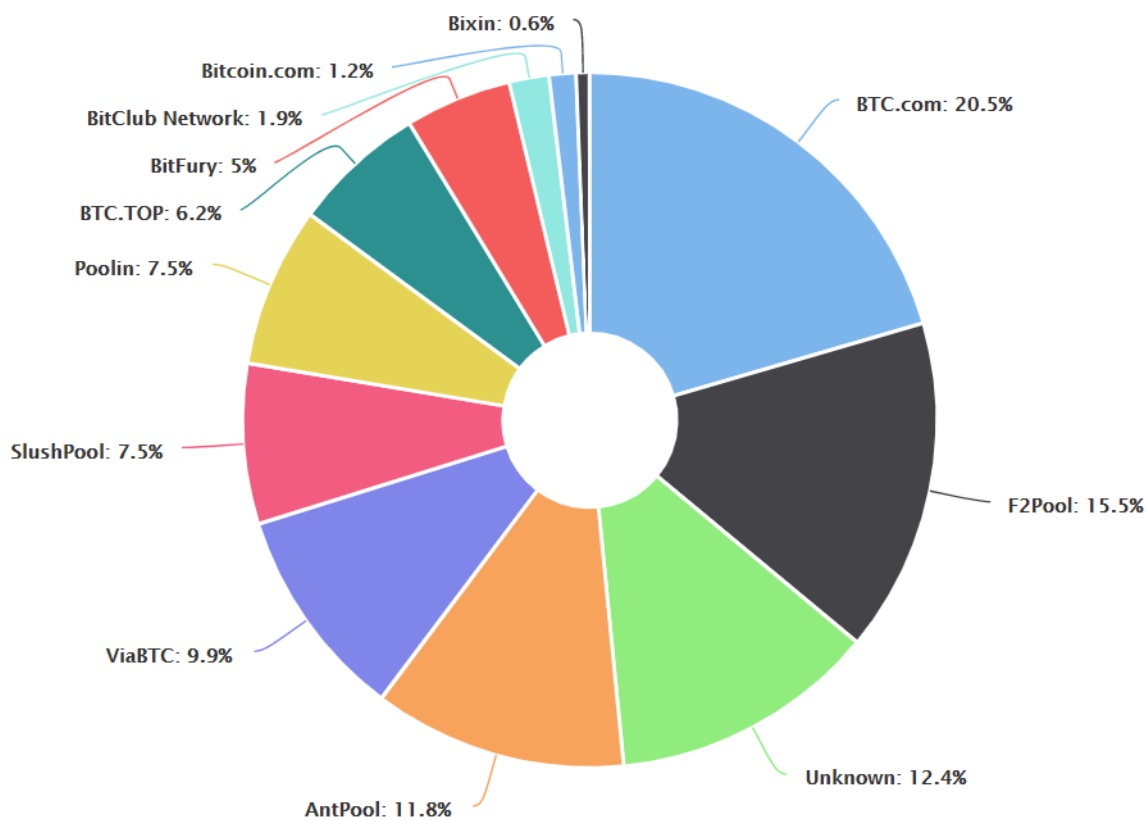


Figure 2.12: Estimation of computing power distribution in Bitcoin amongst the largest mining pools  
Estimation over 24 hours, captured 05/06/2019 on blockchain.com<sup>12</sup>

**Centralization** In the early days of Bitcoin, people were mining with their own computer. But the required computing power rose with the value of bitcoins and soon, casual mining became impossible.



First CPU mining was trumped by GPU mining. Mining farms started appearing: Warehouses full of servers mining non stop. In 2013, specialized hardware called ASIC (Application-Specific Integrated Circuits)<sup>13</sup> was introduced<sup>14</sup>. But this solution requires quite an investment. In addition, this specialized hardware is really good at mining but happens to be bad at everything else. Including keeping up with the blockchain state. So to better their odds, people started organizing themselves into mining pools<sup>15</sup>. Figure 2.12 shows the distribution of computing power amongst the different pools. *Unknown* corresponds to block where the origin of the miner could not be determined. Nowadays, the chances of mining a block when operating outside of a pool are abysmal.

Each pool is organized in a different manner but for the most parts, pool participants share their computing power and divide earnings in proportion to their contribution. This causes several problems. Mainly, it pushes the blockchain towards a re-centralized model based on trust. Wannabe miners need to trust that the person running the pool will actually redistribute the earnings. They also need to trust the owner's intentions.

In the eyes of the network, a pool acts as a single node. If a pool were to aggregate more than 50% of the network computing power, its owner could mount a successful attack on said blockchain (see Section 2.8.4). Because participants in pool not only mine in its name, they also get the block information from it, including the previous block.

Even if no single pool reaches the dreaded 50%, collusion amongst pool owner is a strong possibility. Bitcoin blocks mined in 2018 could be attributed for more than half of them to the four most popular mining pools: 21.7% for BTC.com, 14.4% for AntPool, 11% for Slush, and 10.7% for ViaBTC<sup>16</sup>.

Obviously, pool being a sum of individuals and not one entity, miners are free to leave a pool if they believe it misbehaves or is dangerous to the security of the network as a whole. In Bitcoin history, a single mining pool has reached more than 50% of the network power in three occasions. And every time people left the pool until it was back to an acceptable level. But there is no guarantee this will always be the case, because being part of the biggest pool means more revenue. When bitcoins are not mined by individuals passionate about the new technology but by corporation for instance, this self-regulating behaviour is unlikely to continue.

## 2.6.2 Proof of Stake (PoS)

Proof of Stake has been introduced in an effort to move away from the computationally taxing PoW. There are several variants but the main idea behind it is: the more stakes one holds, the more they have to loose and therefore the more trustworthy they are. The first PoS coin, Peercoin<sup>17</sup>, was released in August 2012.

<sup>13</sup>Bitcoin Wiki: List of Bitcoin ASICs. [https://en.bitcoin.it/wiki/List\\_of\\_Bitcoin\\_mining\\_ASICs](https://en.bitcoin.it/wiki/List_of_Bitcoin_mining_ASICs), Last Checked: July 28th, 2019

<sup>14</sup><https://bitcoinmagazine.com/articles/avalon-ships-bitcoins-first-consumer-asics-1358905223>, Last Checked: July 28th, 2019

<sup>15</sup>Bitcoin wiki: Pooled mining. [https://en.bitcoin.it/wiki/Pooled\\_mining](https://en.bitcoin.it/wiki/Pooled_mining), Last Checked: July 28th, 2019

<sup>16</sup>Information extracted from <https://bitcoinchain.com/pools>, from 31/12/2017 to 31/12/2018

<sup>17</sup>PeerCoin: <https://peercoin.net>, Last Checked: July 28th, 2019

**How it works** In its most basic form, PoS works as a lottery. Each second, each account has a certain probability to be chosen to mine a new block that is proportional to its balance. To mine a block, one must simply sign it. Letting a single participant sign is dangerous as we want a system that a single player cannot dominate. PoS therefore requires  $p$  validators per block. The *lottery* system selects more than  $p$  winners as some winners may not be online at that moment, or the key associated to the selected funds may have been lost.

This method has some advantages over PoW. First, it is more eco-friendly. Secondly, PoS addresses PoW's centralization concerns evoked above. The block generation can be much faster in PoS. The scheme can arguably be considered more secure as a 51% attack (see Section 2.8.4) requires owning 51% of the currency. But it cannot be used in this simplest form as many issues need to be addressed.

**Nothing at stake** Blockchain forks are bad for the network. They introduce uncertainty and negatively affect the value of the currency and the trust of participants in the system. Transactions valid in one might not be in the other and may end up being cancelled, since only one will eventually remain. It is therefore in the network's best interest to limit the number of forks and resolve them rapidly when they arise.

When a fork occurs, nodes must decide which chain to contribute to. With PoW, the best strategy is to mine on the chain that is most likely to win. Dividing computing power between the two chains is not in the miner best interest as it lowers its chances to mine a valid block on either. Furthermore, if the miner chooses the wrong chain, all its efforts are lost. It is also increasingly unlikely that blocks will keep being found at the same rate on both chains. So the fork will rapidly be resolved as one chain becomes longer.

With PoS, voting is costless. The best strategy is therefore to vote on both chains so that whichever chain wins in the end, the reward can be claimed. This is called the *Nothing at Stake* problem. This behaviour causes forks to go unresolved and two alternate chains to exist concurrently for a long period of time.

One might believe that, knowing this behaviour undermines the value of the currency, people with stakes in the system would not be inclined to indulge in it, i.e. they would not vote on both chains in the event of a fork. But it is in every participant's best interest to do so, thus increasing personal gain over the interest of the network as a whole. Hence many participant are likely to do it. This is called the *Tragedy of the Commons*, a situation where individuals acting in their own self-interest cause the ruin of a shared resource.

**Mitigating short range attack** To solve the *Nothing at Stake problem*, Kwon [Kwon, 2014] demands a warranty from the voters. Before being able to vote, participants have to put down a security deposit, coins are then said to be bound. In this case, the voting power no longer depends on the balance of an account but on the amount committed in the security deposit. Every block needs to be validated by at least  $\frac{2}{3}$  of the voters (in actuality, a fraction of voters that holds at least  $\frac{2}{3}$  of the bound coins). A fork therefore demands at least  $\frac{1}{3}$  of the voters to double vote.

Somebody witnessing two blocks of the same height signed by the same participant can publish a transaction with both signatures and expose the voter. If the double voter is caught, the security deposit is lost. The voter having no more committed stakes loses its ability to vote in addition to the amount that

was committed. The participant that produces the evidence might even get the deposited amount. This incentivises good behaviour from the voters but also enforcing of this behaviour by all participants.

Security deposits however, cannot be locked forever. If the funds are never returned, they are as good as lost and no longer act as a deterrent from bad behaviour. The cost of entry would also be prohibitive for many. Therefore, honest voters must be able to recover their deposit after some mandatory lock up time.

Attacks involving a fork can be separated into short and long range attacks. The short range attacks try to fork from a recent block. The PoS as presented by Kwon [Kwon, 2014] punishes this kind of attack. Indeed, if the attack lasts less than the mandatory lock up period of the funds, any attempted attack can be punished in the way described above.

A minimal cost can even be imposed. Let  $n$  be the number of expected participants and  $m$  the minimal amount to be committed by each of them. The total amount of coins committed for this PoS is at least  $n * m$ . A successful attack requires at least  $\frac{1}{3}$  of the overall voting power to double vote. This will result in at least  $\frac{n*m}{3}$  coins being destroyed in the aftermath. By approximating  $n$  and tuning  $m$  accordingly, we can decide on a minimal cost.

But what of the long range attacks, where an alternative chain starts from an old block?

**Weak subjectivity** The problem of double voting is not limited to the resolution of short-range forks. With PoW, creating a block is expensive, so the idea of starting a fork really far back is absurd. In PoS, not so much. Since voting on a block is free, trying to rewrite the blockchain going 1000 or even more blocks in the past, possibly all the way to the genesis block, can work.

In an entry posted on the Ethereum blog [Buterin, 2014], a slight change to the security model is proposed that would solve the problem of long-range attacks: Weak Subjectivity. The author divides consensus algorithms into two categories. The first, the objective algorithms, require new nodes joining the network to be aware of only two things: the protocol definition and the set of all published blocks. With this information alone, one can be aware of the current state of the system. PoW falls into this category.

In the second one, subjective algorithms, different stable states can cohabit and nodes can come to different conclusions. Additional information is required to fully understand the operating of the system. It is for example the case of reputation systems where social information play an important role. The author then proposes a new denomination for PoS, namely weakly subjective. A new node joining the network would require knowledge of the protocol, the set of all published blocks, and additionally a valid state from  $n$  blocks ago. This essentially forbids to go back more than  $n$  blocks. Long-range attacks are no longer possible. The security deposits would have to stay locked up for a minimum of  $n$  blocks during which double voting can be properly punished. Honest voters can retrieve their deposit after the lock-up time. For an attack to be successful, it must last more than  $n$  blocks.

There is however one problem with weak subjectivity: where do new nodes get their stable state from? Nodes that connect at least once every  $n$  blocks are not concerned. But new participants or nodes that have been absent for a while need some external help to update. We therefore need blockchain explorers or other means of bringing people up to date with the system.

**Exponential subjective scoring** In the same entry, another method is proposed under weak subjectivity: exponential subjective scoring. Simply put, each block has a score that depends on that of its parent block but also on when it was first seen. When a node first sees a block from another chain, it penalizes the late comer by assigning it a lower score. How low a score depends on how far in the chain the node needs to go back to find the first common ancestor. The further, the lower. Each node calculates this score for itself. The longest chain is now considered in terms of cumulative scores.

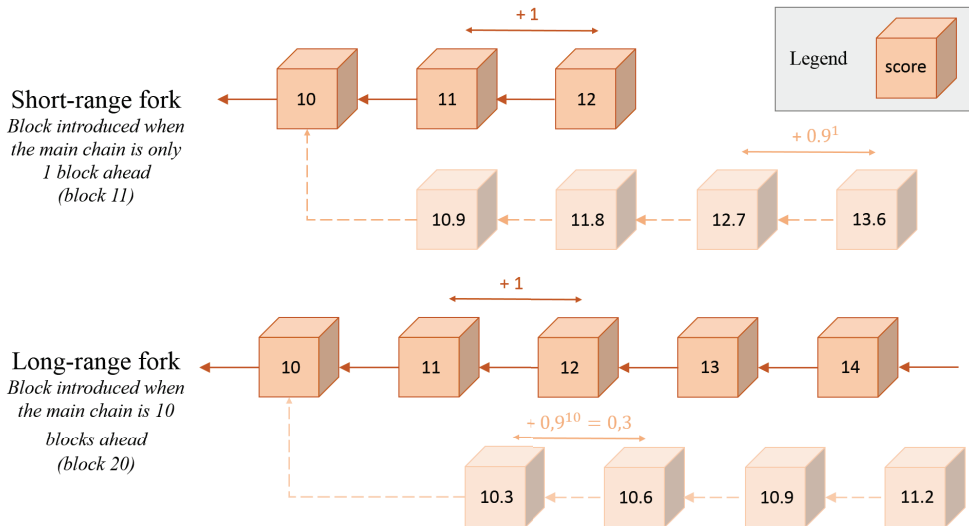


Figure 2.13: An example of fork attempts with exponential subjective scoring

Figure 2.13 provides an example of both short and long range forks under that method. When no fork is occurring, each block's score is calculated by adding 1 to the score of its parent block: the score of the genesis block is 0, it is 1 for the block of height 1, 2 for the block of height 2, etc. When a fork occurs at recent height, the block that is seen second is given a slightly lower score. We take the example of a naturally occurring fork. When the second block arrives, the main chain is only one block ahead (so up to block 11). The score for the new chain is calculated by adding  $0.9^1 = 0.9$  to each new block instead of 1. So the second chain has a slight disadvantage. But as time progresses, the second chain produces block faster than the main chain. After 3 blocks, the second chain has caught up. It becomes the longest chain and the fork is resolved.

In the case of a long range fork however, the difference between the scores of the two chains will grow exponentially, every slight difference adding up to a significant gap in scores. In our example, the second chain starts 10 blocks behind. The score for this chain is computed by adding  $0.9^{10} = 0.3$  to each new block instead of 1. It will therefore take a lot more blocks to catch up to the score of the main chain. And the furthest you start your fork, the harder it will be. This mitigates long-range attacks.

### 2.6.3 Other methods

Neither PoW nor PoS are perfect and there are a number of proposal with varying degrees of maturity that aim to replace them. Here we focus on two of them.

The first, Proof of Activity (PoA), is a hybrid between PoW and PoS. Proposed by Bentov et al. [Bentov et al., 2014], the motivation for PoA comes out of concerns over the attack environment of cryptocurrencies after the original money supply has been distributed and the only reward for block creation comes from transaction fees. In Bitcoin, new coins are introduced by mining new blocks, and the block reward is designed to half periodically until it becomes null. Bentov et al. find that PoW miners might not have a strong enough incentive to maintain the security of the network after that, as coins they mine might be exchanged for other currency and mining hardware can be re-purposed for other cryptocurrencies. Stakeholders are more invested in the system. Hence their proposal.

The second, Ripple's consensus protocol [Schwartz et al., 2014], is centered around behaviour and can be boiled down to a reputation system. The security model it offers is significantly different from PoW or PoS. Ripple is arguably not a blockchain but is a consensus-driven network.

#### 2.6.3.1 Proof of Activity (PoA)

Bentov et al. [Bentov et al., 2014] propose to extend PoW with PoS. The power required to mine a block is much lower than with classic PoW but so is the reward. Because mining a block is not enough, the block then needs to be signed in a PoS fashion. The combination of the two methods is called Proof of Activity (PoA). PoA requires more rounds of communication and more involvement from the miners, hence its name.

Authors identify three ways in which PoW mining will be subject to the *Tragedy of the Commons* after the fixed supply of coin has been mined, and miners solely rely on transaction fees for payment. The *Tragedy of the Commons* refers to a situation where individuals acting in their own self-interest cause the ruin of a shared resource.

The first concerns the willingness of users to pay transaction fees. Even if everybody agrees they must be paid to incentivize miners, it is in everyone best interest not to pay too high a fee themselves. The solution to this is to not include transactions with low fees in mined blocks. Which brings us to the second contradiction.

It is in the miners best interest to agree on a minimum fee. If they did, people would have to pay or their transactions would never be included. However, miners that do not follow the rules will increase their profit as they will include more transactions in their blocks. A proposed solution is to add some protocol-enforced rules such as capping the number of transactions per block. This way, miners only include the transactions with the highest fees which would force users to pay them. The system must remain attractive to miners as the blockchain security level is directly dependent on the number of miners participating in the consensus protocol. But this cost might be more than the users are willing to pay.

The last problem is that the fees are only paid to the miner that created the block. But the cost of running the network are also in propagation, verification, and storage and are shared amongst all the nodes. And these operations are not being compensated. This could encourage miners to stop propagating the transactions to keep the reward for themselves, a problem is discussed in Section 2.8.7 A solution

would be to limit the allowed size of a block, that way miners cannot possibly include all their withheld transactions in one block, especially the longer ones that tend to come with a higher fee.

The protocol involves two types of actors: miners and signers. Miners produce empty PoW blocks (no transactions included). The empty block is used to deterministically derive  $n$  pseudo-random values. A special function called *follow-the-satoshi* uses these pseudo-random values to select  $n$  satoshis<sup>18</sup> amongst all the coins created from the genesis block to the most current one. It does so by assigning a number to each satoshi, starting from 0 for the first coin ever mined. The pseudo-random values are normalized to fit inside the interval and point to a specific coin. The life of the coin is traced from origin to the hand of its current owner. Each satoshi selects one signer.

The first  $n - 1$  signers simply sign the PoW block. The  $n^{\text{th}}$  one creates the final block by including the PoW block, transactions, signatures from previous signers, and finally their own. The longest chain is still considered in terms of PoW. The fees are divided unequally among the miner, the signers and the final block creator. PoW is used mainly to slow down the block creation process. It is a reward system rather than a punitive one. It incentivizes people to run full nodes and participate actively in the network.

### 2.6.3.2 Ripple

Ripple's consensus [Schwartz et al., 2014] is somewhat different from what we have seen and involves more trust. The network is divided into Unique Node Lists (UNL). Each participant has their own set of nodes that they trust collectively to reach consensus. During the consensus process, a node will only interact with and consider transactions proposed by members of its own UNL. The number of nodes in each participant UNL must be such that the sets have a non void intersection (to avoid a fork). This intersection must be at least a fifth of the size of the bigger set.

The set of transactions the network agrees on is stored in the *Last Closed Ledger*. Each node maintains its own *Open Ledger* that contains unconfirmed transactions. The process is comprised of several rounds, each containing the following steps:

1. **Proposal** - Each node assembles the valid transactions that have not been validated by the network yet, constitutes a candidate set, and makes it public.
2. **Vote** - Each node collects the candidate sets of all nodes in its UNL and votes on whether to accept each transaction contained within.
3. **Results** - Transactions that are deemed valid by more than a certain percentage of the voters continue to the next round while the others are dropped. They can be proposed again in the next consensus iteration. The percentage of positive vote needed to move to the next round is supposed to increase from round to round. The number of rounds is not fixed by the protocol. However, the final round must require an acceptance level of 80%.

This protocol, when the UNL sets composition verifies some properties, is resistant to up to  $\frac{n-1}{5}$  dishonest or faulty participants, where  $n$  is the total number of participants. This bound is lower than some other protocols but its rapid convergence and flexibility are properties that other protocols lack.

---

<sup>18</sup>The satoshi is the lowest division of the Bitcoin. 1BTC =  $10^8$  satoshis

Ripple also verifies three important properties. *Correctness* guarantees that the system can differentiate between valid and invalid transactions. For instance, a transaction from Belinda transferring 5 dollars to Camille will not be accepted if Belinda's account does not hold these 5 dollars.

*Agreement* guarantees that all nodes in the system have the same history and state for it. Among other things, it prevents double spending (see Section 2.8.1). Belinda's account holds only 5 dollars and she tries to transfer 5 dollars to both Camille and Devlin. Each transaction taken alone is valid as Belinda's balance allows for the transfer. But both cannot be applied. This is why agreement is needed on which transaction to apply in what order.

*Utility* can be defined as the usefulness of the system. As with a user interface, responsiveness is key. A system that is sure to reach consensus but does so in an unpredictable or a very long amount of time will never find real life application. Utility here is therefore taken to mean that the consensus is reached rather quickly. All three properties hold only when the number of malicious nodes are beyond the above mentioned bound.

Each node can decide to kick another node out of its UNL. A node that keeps proposing invalid transactions or consistently votes to discard all transactions for example would be dropped. Nodes that are too low and slow down the consensus process can also be dropped to ensure utility. It is in that sense that the Ripple Consensus Protocol leverages behaviour.

## 2.7 Governance

Blockchains can assume several types of governance. As we have seen, Bitcoin and Ethereum are resolutely open, allowing anybody to join at any time. Applications outside of cryptocurrencies however run into some liability and privacy issues that require a different approach. Hence permissioned blockchains, such as Hyperledger Fabric, where the rights of the participants can be restricted.

Even beyond that classic divide, the blockchain governance is a complicated subject. Who writes the original code? Who owns it? Who decides on an update to the software or to the protocol? Can we correct a mistake once the blockchain has been launched? Does everybody's voice truly carry the same weight in debate? What is the cost of openness on maintainability? Many questions that we explore below, after presenting the differences between public and permissioned blockchain.

### 2.7.1 Public vs Permissioned

Regardless of the permission model, a blockchain is always a distributed network which participants maintain a shared, append-only ledger. Transactions are ordered through a consensus protocol. The network provides some guarantees of immutability despite faulty or malicious nodes. Where they differ is in who is allowed to participate in the network and in what ways.

**Public blockchains** In a public blockchain, all operations are available to all users. There is no need for registration. Identities can be generated by the participants themselves. Users can join the network dynamically. No access control need to be implemented.

The fundamental network rules are defined by the community and can only be changed if the community agrees. No one can unilaterally impose revisions to them. There is no oversight on what can and

cannot be part of the ledger (except if mentioned explicitly in the rules). Transactions cannot be censored unless it is the wish of the community at large.

A public blockchain also enables cost sharing amongst individuals, giving them access to a higher level of security than they could achieve on their own.

On the flip side, miners need to be incentivized to secure the network. The more miners participate in the consensus, the more secure the blockchain. But all these people are not gonna spend money and energy to do so out of the goodness of their hearts. Hence the cryptocurrency to back up the blockchain and reward the miners. But, the more miners, the more computationally taxing or complex the consensus protocol.

Finally, this openness leads to privacy compromises. But not every use case requires privacy. Let us consider the parliament of an imaginary country where new laws are voted using the blockchain. In this instance, the vote of every representative should be public knowledge as people are allowed to know how their representative votes on each issue. A smart contract tallies up the votes, ensuring transparency in the matter. The code for this smart contract is publicly available and verifiable. That way, no one can vote for absent colleagues, manipulate the results, change or bury the record of their individual vote after the fact, etc.

It is important to note that even on a public blockchain, permission restriction can be enacted: only the intended recipient can spend a bitcoin, smart contracts can be coded to only accept calls from the owner's address, etc. But there are cases where this is not enough and a public blockchain is not suited for the job.

**Permissioned blockchains** In a permissioned setting, some operations such as mining a block, publishing a transaction, or even reading the blockchain can be subject to permissions. Hyperledger Fabric is an example of such a blockchain. In order to perform a restricted operation, users need to register, authenticate, and be granted privileges. An access control system must be deployed. A small number of individuals possess the power to change the rules of operation which can lead to censorship.

As participants are vetted before joining the network, security can be more easily achieved. Furthermore, participants have built-in interest and may not require an incentive to participate in the consensus protocol. However, the re-centralization of power can lead to an arguably less secure model.

We differentiate between two types of permissioned blockchains : private blockchains, and consortium blockchains.

*Private blockchains* are run by a single entity that can unilaterally enact changes to the system. An example for such a blockchain can be a large company that wants to track the supply chain from its subcontractors. The company sets the rules and can for instance decide that a contractor is no longer able to participate in the network after its contract has ended. The blockchain can help the company track its supply all the way down the chain of subcontractors. Records can be open to outside audit agencies as the need arises.

*Consortium blockchains* are run by a small number of entities. They may have different stakes in the system and its activity, but they all decide on how it operates. Let us take the example of electricity companies operating over different geographical locations. These companies want to buy and sell electricity to one another. Electricity produced by themselves but also by individuals residing within their domain. No one is willing to assume the cost and the liability of operating the shared system. A consortium



blockchain answers this conundrum. The companies can decide if they will all have equal voting power, under what conditions the blockchain can be modified, and so on. Once they agree, individuals can post their production on the blockchain for every one to bid on.

### 2.7.2 Governance models

This section applies mostly to public blockchains and consortium blockchains to a lesser degree. Private blockchains are controlled by a single entity and can decide to unilaterally enact changes. For the others, governance is a delicate subject.

Over the course of its life, a blockchain development faces three kinds of challenges:

1. Fixing bugs and vulnerabilities,
2. Thwarting and ultimately repairing damages from attacks,
3. Upgrading.

A blockchain ecosystem is made up of three types of actors: Network nodes (including miners) keep the system running, users are the reason the system exists, and developers tend to the system. Additionally, the value of a cryptocurrency has a direct impact on its blockchain's ecosystem. Exchange platforms can therefore be included as a fourth external actor. These actors have different interests. Yet they must agree on how the blockchain should work.

When operated properly, a governance system should ensure reactivity against attacks, continued involvement of the different actors, and competitiveness over other applications. One of the appeal of the blockchain paradigm is its decentralized nature. Nowadays, users are being dictated terms and conditions that they cannot argue. The blockchain can be a way for them to claim back that power and shape applications that fit their needs and beliefs. Which is why governance is so fundamental to blockchain applications. The upgrade proposal process should also be as intuitive as possible as to no hinder creativity and innovation in developpers.

Above all, the governance process should be transparent with thresholds and requirements clearly stated: the vote is only valid if at least 35% of actors participate, this proposal requires the approval of the lead developer to go through, a proposal should be backed by at least 50 people before it can be brought up for discussion, and so on. Rules and guidelines should include a process to modify the governance system itself.

When devising a blockchain governance system, two choices can be considered: off or on the blockchain.

**Off-chain Governance** In this first model, an external platform is required to debate system modification. Bitcoin and Ethereum have both opted for that option. In both cases, Improvement Proposals (BIP<sup>19</sup> and EIP<sup>20</sup>) are suggested by groups or individuals and then debated. If accepted, they are integrated to softwares and protocols.

---

<sup>19</sup><https://github.com/bitcoin/bips>, Last checked: July 8th, 2019

<sup>20</sup><https://github.com/ethereum/EIPs>, Last checked: July 8th, 2019

This method is susceptible to centralization as some actors hold more sway in the ecosystem than others. In Ethereum for instance, the Ethereum Foundation has been accused of being too present in the debates. In Bitcoin, mining pools and core developers have a disproportionate impact on decisions. Some actors may also be restricted by their technological know how or financial means.

Off-chain governance is slow by nature as a consensus needs to be reached by social means. It has however worked nicely in the past. When a change has to be enacted, users have to actively opt in by upgrading their blockchain clients or changing their configuration. As such users hold the cards in the end.

**On-chain Governance** On-chain governance is a more recent development. There is not enough hindsight to draw any meaningful conclusions yet. On principle though, on-chain governance uses smart contracts to automate governance processes, thus implementing direct democracy.

Coming up with a code that is bug free and implements an efficient governance system tailored to one's need is a gargantuan task. Let alone on a first try. Smart contracts have shown time and time again that the more complex they are, the more likely it is that they can be abused. By nature, they cannot be modified. This is therefore a dangerous thing to automate. Roll back processes should be put in place as we are in an exploratory phase at the moment.

The fear with the *one token, one vote* system that usually goes along with on-chain governance is the disproportionate power that is given to the richest in the community. This replicates existing systems that have left many disenfranchised.

Finally, in this model, upgrades are passive. This lessens the power of blockchain users as well as their incentives to be active participants in the process.

**The case of the DAO** An excellent exercise in blockchain governance is the infamous DAO incident. The DAO (Decentralized Autonomous Organization) was an Ethereum-based venture which goal was to use the blockchain to enable a direct and transparent management of its funds by the investors. No board of directors, no manager, just a smart contract. The DAO was crowdfunded in May 2016 and collected more than 150 million USD from over 11 000 investors<sup>21</sup>, collecting just shy of 14% of the total number of ether issued at the time.

Its code had been publicly posted and reviewed. Some vulnerabilities were discovered and investors were urged to hold off until they were resolved. However, in June of 2016, a vulnerability in the code was used to siphon the equivalent of 3,6 million USD, around a third of the funds stored in the DAO smart contract at the time.

This situation caused a fundamental debate in the Ethereum community. Some consider the hack to be valid behaviour as it operated within the rules. They argue that the blockchain immutability comes first. Others suggested to rollback the clock and re appropriate the funds. This establishes a precedent of the community collectively rewriting history. Others yet called for the dissolution of the DAO. In the end, this resulted in a network split, also known as a fork.

---

<sup>21</sup><https://www.americanbanker.com/news/the-dao-might-be-groundbreaking-but-is-it-legal>, Last checked: June 11th, 2019

### 2.7.3 Forks

Regardless of the governance model, updating a blockchain presents some challenges. A blockchain is always live. Rebooting it is contrary to its seminal principle of persistence.

A modification to blockchain rules causes a fork in the system: until it is resolved, nodes are working with different sets of rules. Transactions that are valid for one, may not be for another. This splits and weakens the system until the dust settles and the fork can be resolved.

The forks we mention here are different from the one described in Section 2.3.6. They are mostly voluntary and pertain to protocol changes. The forks of Section 2.3.6 occur naturally and are accidental. In this latter case, the choice of which chain to continue is mostly inconsequential whereas in the former case, the success of the fork has deep implications for the development of its blockchain.

Regardless of the debates that preceded it, a fork's success depends on its adoption. Two kinds of nodes are particularly influential in this matter: miners and full nodes.

Nodes in the network can choose to maintain a local version of the whole blockchain, or simply keep the parts that interest them and query the rest when necessary. The former are called full nodes. Only full nodes can verify transactions as they have the entire history at their disposal to compare them to. Users that do not run a full node rely on them to access blockchain history.

Miners can be full nodes themselves or trust another node to provide them with valid transactions to bundle in their blocks. Likewise, full nodes can be miners or lack the computing power (or fund, or interest) to join in the consensus protocol.

Forks can be soft, or hard, depending on the compatibility of the new network with the old one. We present both cases below.

**Soft fork** After a soft fork, blocks produced under the new rules are still considered valid by the old software. In that, a soft fork is backward compatible. Old blocks however are considered invalid by the new system. Or rather only a subset of previously valid transactions remain so.

An example of Bitcoin soft fork would be *Pay to Script Hash (P2SH)*. Prior to this, Bitcoin transactions were sent to an address hash that simply specified the recipient. With P2SH, users can specify a script instead. As detailed in Section 2.5.1.3, the recipient must then provide a script matching the hash, as well as data that will make it evaluate to true in order to unlock their funds.

A soft fork can be activated either by miners that will start enforcing the new rules, or by full nodes that will drop block including newly invalid transactions. It is not in the interest of miners to have their blocks dropped as they lose out on the block reward. Miners therefore have an incentive to fall in line with the full nodes.

For a soft fork to be successful, half of the miners (or half of the power behind the consensus) need to switch to the new rules. Miners that do not make the switch weaken the network for two reasons: First, this lowers the total power behind the consensus as blocks produced by these miners will be dropped by the majority and therefore not make it into the blockchain, effectively removing these miners and their resources from the block creation race. Second, this opens the network to the *fake confirmation* vulnerability.

To illustrate this attack, let us consider a malicious character  $M$  and an honest victim  $V$ .  $M$  produces a transaction transferring money to  $V$ . The transaction is purposefully not new-rule-compliant. It is

however valid for pre-fork miners and ends up in a block. This block will not be accepted by the network at large. More than half of the miners deem it invalid and will not build on top of it. However some full nodes that are yet to switch over may add it to their chain.  $V$  can then be tricked into thinking that the transaction has been accepted.

Soft forks can also be used to undo accidental hard forks.

**Hard fork** After a hard fork, new blocks are considered invalid by the old software. This essentially creates a new chain and splits the network forever. Out of the two chains, the winner will be the one adopted by the majority. This majority is defined in terms of miners and full nodes of course, but also users, developers, and even cryptocurrency exchange platforms.

In 2016, the Ethereum blockchain operated a hard fork to undo the DAO hack. Since then, two chains have co-existed: the main chain that is still maintained by the Ethereum foundation and remains the official carrier of the ETH currency accepted to undo the hack. Those who didn't keep with the original chain and created a new currency called Ethereum Classic (ETC). At the time of writing<sup>22</sup>, ETC is valued at \$8,25 against \$245,46 for ETH.

Readers interested in the history of blockchain forks are directed to an article by McCorry et al. [[McCorry et al., 2017](#)].

## 2.8 Blockchain security: attacks

In this section we discuss blockchain vulnerabilities. Vulnerabilities can stem from the blockchain concept itself, each blockchain's own protocol, or client implementation. We address all three aspects, though the security and limitations of consensus protocol have been addressed in Section 2.6.

### 2.8.1 Double spending

Great efforts have gone into making paper money difficult to forge: type of paper, design, watermarks, serial numbers, etc. In a digital setting where tokens can be easily replicated, we face the same issue: How do we prevent users from duplicating their tokens and using them multiple times. Figure 2.14 illustrates this process called *Double Spending*. It is the main ordeal digital currency must overcome in order to be usable. Indeed, if double spending is possible in your system, when you have any amount of money, you have an infinite amount of it, rendering the whole system meaningless. Outside of cryptocurrency, double spending can be interpreted as reversing any type of transaction.

In centralized settings, tokens must be presented to the *bank* that checks their validity and records transactions. This requires trust, creates a bottleneck, and a single point of failure.

The blockchain did away with central authorities by making all transactions public so that everyone can check for themselves whether a token has already been spent. In this setting, if not trivial, double spending is still a threat. We detail two ways they can happen and how to protect against it.

---

<sup>22</sup>June 11th, 2019

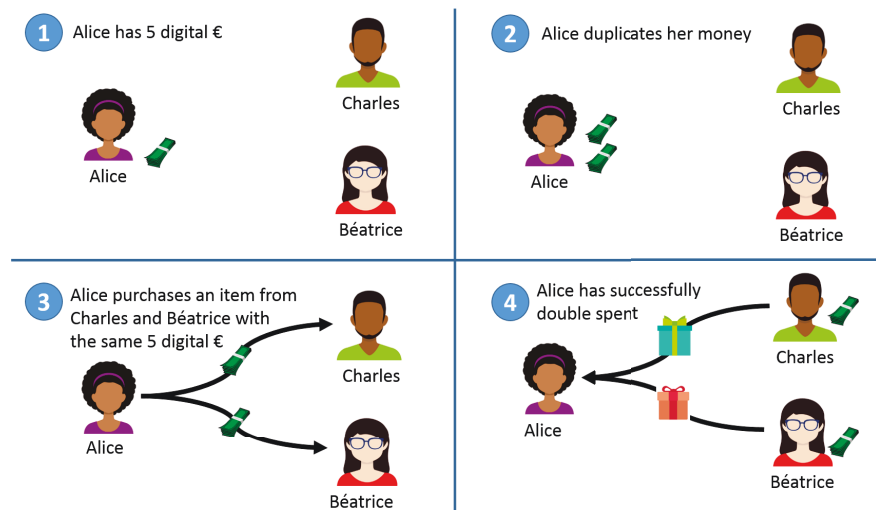


Figure 2.14: Double Spending

**Unconfirmed transactions** As seen in Figure 2.7 of Section 2.3, transactions are issued, broadcasted, and validated before being included in a block. The delay between transaction issuance, validation, and integration into the blockchain can be pretty long. A transaction that has been validated but is yet to be included in a block is called an unconfirmed transaction. One could expect that all valid transactions will eventually make their way into the blockchain. It is not the case.

When one wants to pay for a direct service, such as a cup of coffee, or a hair cut, it becomes tempting to accept unconfirmed transactions as the client is unlikely to sit there until the transaction goes through. This opens up a double spending opportunity: Placing ourselves in the Bitcoin model, imagine a client paying for a fancy haircut. They issue a blockchain transaction to the hairdresser with a low transaction fee. The transaction is validated by full nodes and the client goes on its merry way. As soon as they leave the salon however, the client issues a transaction to themselves, using the same UTXO as inputs, with a higher fee. The first transaction is not yet part of the blockchain. This second transaction is therefore considered valid by full nodes. A block cannot contain both transactions however and miners are incentivized to choose the transaction with the highest fee to maximize their profit. The second transaction is more likely to win and eventually cancel the other one.

Recipient should never accept an unconfirmed transaction as proof of payment. But confirmation is not enough.

**Naturally occurring forks** We have seen in Section 2.3.6 that blockchain forks were a naturally occurring phenomenon. In this event and for a brief moment, the network is split. This is another opportunity for a double spend.

Maleficent issues two transactions  $tx_A$  and  $tx_B$  to Alexander and Belinda respectively. The transactions are such that they contradict one another and cannot coexist. She sends them to the network

through different nodes, ideally on opposite sides of the network so that roughly half of the network sees  $tx_A$  first, and the other half receives  $tx_B$  first. Now, in the event of a fork, chances are that candidate block  $B_1$  includes  $tx_A$  and candidate block  $B_2$  includes  $tx_B$ . Maleficient can now produce a valid block including  $tx_A$  to Alexander and do the same for Belinda.

Naturally occurring forks cannot be predicted. Their rates varies depending on the blockchain. The chances of this attack being successful are somewhat slim. A natural fork will resolve itself over time. To protect against this, users are encouraged to only consider a transaction as definitively embedded in the blockchain if the block that contains it is buried under at least  $n$  blocks, where  $n$  is blockchain dependent. For Bitcoin,  $n = 6$ .

Consider a transaction  $tx_0$  embedded in block  $B_i$ . If an attacker can produce an alternative chain starting from block  $B_{i-1}$  that is both valid and longer than the main chain,  $tx_0$  would be reversed. The chances of success depend on the underlying consensus protocol (see Section 2.6). In general, the longer the alternative chain, the slimmer the chances.

The odds change when the attacker takes over the consensus process. This case is treated in Section 2.8.4.

## 2.8.2 Key security

On the blockchain, one's private key is one's identity. If a key is accidentally lost, the associated funds cannot be recovered, smart contracts cannot be accessed, there is no built-in insurance or protection mechanism. If a key is stolen and an illicit transaction is accepted, that transaction cannot be reversed. This is the pendant of decentralization: there are no authority to mitigate conflicts and protect users.

Key security is therefore central to blockchain security. One approach is to focus on one private key and deploy heavy protection mechanisms. Another is to mitigate the risks by lowering your dependency on any single key.

**Hot and cold storage** Private keys can be stored either on a support that has Internet access, this is called a hot storage, or on a support that does not, which is called a cold storage. The choice is one of security versus usability.

Hot storage options are evidently less secure as their Internet access exposes them to any and all attacks. They are recommended for keys that are used regularly. Taking the case of a smart meter reporting a consumer's electricity consumption to a smart contract set up by the electricity company, only hot storage will do. Otherwise, an operator would need to physically plug and unplug the key each time a report is to be sent out.

Cold storage options sacrifice usability for security. They are suited for key that are barely used or which security is paramount. Taking the same use case, the key that deployed the smart contract in question and that is the only one allowed to invoke administrative functions or modify the contract should be stored in a cold storage.

**Wallets** As evoked in Section 2.5.1.2, wallets can be used to store private keys. Aptly named, a hot wallet should only be used to unlock small amounts or invoke limited functions. This is not where you keep your savings or your master key.

Cloud and other online wallets should be avoided as they make a juicy target for attackers and do not necessarily have the security to match. A wallet should always be backed to protect against human or computer failure. Multi-factor authentication is available to increase the security level. Patches should always be applied as many blockchain hacks do not target the protocol but wallet implementations.

Multi-signature wallets are an elegant solution to protect against theft. They can be used to protect an important company account (3 out of 5 accountants must sign off on a purchase), or to split one's keys on multiple devices such as one on a computer and one on the phone.

**One key, one use** A long standing adage in cryptography is "*One key, one use*", meaning that you should use different keys for different purposes. This is especially true on the blockchain. Funds should be split amongst different addresses, different keys used to deploy and administer different smart contracts, etc.

### 2.8.3 Sybil attacks

On the blockchain, every one can have as many identities as they'd like and run as many nodes as they'd like. A Sybil attack is defined by an individual creating numerous identities that appear to be disconnected for nefarious goals. Blockchain consensus protocols are built to resist Sybil attacks and rest on costly resources that cannot be artificially inflated. The blockchain network however is not immune.

**The importance of full nodes** The blockchain has a number of systems in place to prevent tampering. Those systems are only efficient if users take advantage of them. For instance, assume that an attacker modifies block  $n$  by removing a transaction. The hash of the block no longer corresponds to the value stored in block  $n + 1$ . However, if users do no check for discrepancies, they can be easily abused.

Some users don't have the necessary resources to run a full node. They simply request information from those who do. This makes them vulnerable to Sybil attacks. An attacker can pose as a legitimate full node and send them false information. To take advantage of the distributed nature of the system, users should always cross check information obtained from several nodes, preferably including a few that they trust.

**Isolating a target** An attacker can attempt to flood the network with their own nodes in order to isolate their target. Once the target is only connected to the attacker's nodes, the attacker is able to shape the information that reaches its target. This can be used as a stepping stone from other attacks. If successful, an attacker can:

- block out every block or transaction, effectively disconnecting the target from the network,
- only relay their own blocks, effectively putting the target on a separate blockchain controlled by them where it is trivial to mount a double spending attack,
- censor blocks and transactions, which also facilitates a double spending attack.

Additionally, the attacker can analyze the transactions sent from the target's node, thus breaking pseudonymity by identifying them as the sender. Observing the rate at which blocks are created can help spot the second attack. Another way to protect against them is for a user to ensure the IPs they are connecting to are nicely spread out.

### 2.8.4 The 51% attack

In Bitcoin, generating a valid block requires a lot of computation. The longest chain therefore represents the decision of the majority since the most work went into it. Honest nodes will work on extending it while malicious nodes try to create another version of history. Nakamoto [Nakamoto, 2008] claims this alternative version of events will not catch up to the longest chain unless the attackers hold more than 50% of the computing power of the network. This is called the 51% attack. If such an attack was to be successful, its author would be able to:

1. **Only acknowledge its own blocks** - With more than half of the power of the network, the attacker will mine most of the blocks. If the attacker is not the only one mining, it may happen that someone else creates a valid block. The attacker can simply ignore it, refusing to mine on top of any block that isn't hers. Since she mines more efficiently, her chain will eventually be the longest and win in the end. This is the essence of this attack as it allows the attacker to control history.
2. **Monopolize block reward** - A natural consequence of the previous point and even a motivation for it is that the attacker can keep all the block rewards for herself.
3. **Censor transactions**: Since she mines the majority of the blocks, the attacker can decide which transactions to include, and also which to leave out. This can cause delay in the confirmation of a transaction or even cause a transaction to not be included at all. This attack however is visible to the network. Controlling the process of block creation does not allow the attacker to prevent the propagation of valid transactions before they are included into a block.
4. **Reverse transactions** - This last point is at the heart of what digital currencies try to prevent: successful double spending. The attacker wants to purchase an item. She sends a transaction with  $tx_0$  as input that is included in the blockchain as part of the block  $B$  of height  $h$ . She then waits a few blocks for the transaction to be confirmed and for the item to be exchanged. Finally, she starts a new chain from  $B$ 's parent block, so at height  $h - 1$ , that doesn't include the previous transaction and may even include a transaction from herself to herself using  $tx_0$  as an input. This will ensure that the previous transaction can never be included in the blockchain again since its input has already been used, rendering it invalid. Once this chain becomes the longest one, the whole network will switch to it. The transaction has effectively been reversed and the attacker disposes of the same amount to be spent once more.

Such an attack would have terrible consequences on the trust people put in the system. But it does not give an attacker all powers. A successful attack would not allow an attacker to:



1. **Forge fake transactions** - A transaction will not be considered valid unless signed with the private key corresponding to the public key of the owner of the input. Without it, the network will not accept the transaction. Producing a valid signature means either having access to the private key of the owner of the coins or being able to break the underlying cryptographic primitives. None of this is implied by owning 51% of the network computing power.
2. **Steal coins** - Coins are really only transaction outputs. Stealing a coin therefore means producing a transaction that takes said coins as input and has the attacker as the recipient of the output. Since transactions cannot be forged, coins cannot be stolen.
3. **Change the block reward** - The amount of coins a miner is rewarded with is part of the protocol and is shared by everyone. The attacker cannot change the protocol.

The 51% attack is not limited to Bitcoin. All blockchains are subject to it, regardless of their block creation system. All blockchains' consensus processes are based on the holding of a resource, be it computation power or stakes. Holding more than half of the total amount available in the network allows for such an attack. This attack is not only theoretical, smaller crypto currencies have effectively been attacked<sup>23</sup>, including Ethereum Classic, the post DAO spin off of Ethereum<sup>24</sup>.

### 2.8.5 Code vulnerabilities

Vulnerabilities arise not only from blockchain protocols but also from the software that implements them. No code is without bug. Blockchain wallets in particular have been at the heart of several "*blockchain hacks*" and mishaps.

**Wallet hacks** In November 2017, around 280 millions ETH were frozen when an attacker poking around triggered a bug in the wallet software<sup>25</sup>.

In August 2018, Bitfi, an "*unhackable*" wallet retracted their claim after researchers Saleem Rashid and Ryan Castellucci ran an exploit that allowed funds to be siphoned from the wallet, claiming that the flaw could not be fixed with a firmware update<sup>26</sup>.

In September of the same year, Monero announced that their official chrome extension was compromised and allowed attackers to extract keys and password for other websites<sup>27</sup>.

In June 2019, Komodo preemptively exploited a vulnerability in its own wallets to secure at-risk funds from potential attacks<sup>28</sup>.

---

<sup>23</sup><http://bitcoinist.net/hackers-holding-small-blockchains-hostage-by-51-attacking-them/>, Last checked: July 8th, 2019

<sup>24</sup><https://www.theverge.com/2019/1/9/18174407/ethereum-classic-hack-51-percent-attack-double-spend-crypto>, Last checked: July 8th, 2019

<sup>25</sup><https://btcmanager.com/ethereum-wallets-hacked-not-buggy/?q=/ethereum-wallets-hacked-not-buggy>, Last checked: July 8th, 2019

<sup>26</sup><https://ambcrypto.com/john-mcafee-bitfi-wallet-hacked-to-remove-unhackable-tag-from-marketing/>, Last checked: July 8th, 2019

<sup>27</sup><https://ambcrypto.com/monero-xmr-wallets-compromised-as-hackers-target-mega-chrome-extension/>, Last checked: July 8th, 2019

<sup>28</sup><https://btcmanager.com/komodo-wallet-hacks-secure-13-million/?q=/komodo-wallet->

**Security patch** Wallets are not the only one with flaws. Client software, the thing that allows users to connect to the blockchain and run a node, has also shown signs of weakness. The developer community is usually quick to react to vulnerability disclosures and put out a patch. But blockchain users are no different from the users in more classical ecosystem and patches are not being applied, leaving the system vulnerable.

In February 2019 for instance, a DoS vulnerability in Parity (an Ethereum client) was discovered by a team at the Security Research Labs. More than a month later, a third of the nodes had yet to be patched<sup>29</sup>.

**Smart contracts** Smart contracts are another type of code running around the blockchain ecosystem. They are made to be immutable with publicly available code that will be scrutinized. This makes coding mistakes particularly unforgiving. The DAO (see Section 2.7.2) is a good example of an attacker exploiting smart contract vulnerabilities.

A possibility is to formally prove smart contracts before deploying them. This can be a lengthy process but should remove some of the vulnerabilities. Note nonetheless that a formal proof simply ensures that the code is behaving as it should. It does not mean that the way it is behaving is without flaws.

### 2.8.6 Denial of Service (DoS)

Denial of Service attacks flood networks or servers with a high volume of traffic, rendering them unable to operate normally. All networks have to deal with the threat of DoS attacks. Every blockchain has built-in mechanisms to counter them. These fail-safe are implemented at two levels: protocol and implementation.

A list of DoS mitigation in Bitcoin can be found on its wiki<sup>30</sup>. A few of them are:

- Double spent transactions are not forwarded
- A block is only forwarded once per peer
- Misbehaving IP are temporarily banned
- There is a limit to the acceptable script size
- There is a size limit for block

In Ethereum, as we have already mentioned, the cost of operations can vary to regulate traffic. Furthermore, there is a limit on the total amount of gas that can be consumed per block, thus limiting the amount per transaction. This limit can be slowly raised but should prevent malicious users from running infinite loops.

DoS attacks have certainly happened in the past<sup>31</sup> and will again in the future.

---

hacks-secure-13-million, Last checked: July 8th, 2019

<sup>29</sup>[https://srlabs.de/bites/blockchain\\_patch\\_gap/](https://srlabs.de/bites/blockchain_patch_gap/), Last checked: July 8th, 2019

<sup>30</sup><https://en.bitcoin.it/wiki/Weaknesses>, Last checked: July 8th, 2019

<sup>31</sup><https://cointelegraph.com/news/ethereum-is-under-ddos-attack-miners-are-alerted>, Last checked: July 8th, 2019

## 2.8.7 Withholding attacks

An attacker can gain advantages over the other participants by withholding information. This class of attacks benefits from a well connected node that can intercept blocks and transactions and broadcast efficiently their own blocks. Withholding attacks are of two kinds.

### 2.8.7.1 Withholding transactions

Transactions sometimes include transaction fees. When creating a block, these fees are added to the block reward. In Bitcoin, the decrease in block reward pushes the miners to rely more heavily on transaction fees to sustain their efforts.

The number of transactions in a block is limited. Users can include a higher fee to encourage miners to include their transaction over less lucrative ones. But another behaviour can be for miners to not forward transactions with a high fee. That way, fewer nodes know about the transaction and it is less likely to be included in a block by someone else. This leaves the miner in question plenty of time to craft the block with the highest fee possible and maximize profit. This is the equivalent of making sure that the price is at its highest when playing the lottery.

### 2.8.7.2 Withholding blocks

Eyal et al. [Eyal and Sirer, 2014] exposed a strategy they call *Selfish Mining*. Its effect is to allow a colluding minority organized in a pool to produce blocks more often than they should be able to given the fraction of the computing power they hold. The idea is to force the honest majority to waste computation power on blocks that will not end up in the chain.

Because of propagation delays, finding a block gives a tiny advantage over the other participants as one can start working on the next block with a small advance. Pushing this reasoning, keeping blocks private can lead to an even bigger advantage. By holding on to the mined blocks, the colluding minority wastes other miner's resources on a block that has already been solved. This attack of course requires that the colluding minority is able to mine at least one block before it can be launched.

Let  $m$  be the colluding minority and  $M$  be the honest majority. Both sides are working on finding the block of height  $h$ .  $m$  finds the block at time  $t$  and holds on to it. At time  $t + 1$ ,  $m$  starts working on the block of height  $h + 1$  while  $M$  is still working on block  $h$ . This continues until  $t + 5$  at which point  $m$  releases their block  $h$ . From  $t + 1$  to  $t + 5$ ,  $M$  has been wasting their computation power. This attack therefore lowers the effective computing power of the network. The loss only affects  $M$ , the honest majority.

There is a chance that  $M$  finds a valid block between  $t + 1$  and  $t + 5$ . In this case, a Sybil attack can increase the odds of  $m$ : By spawning numerous nodes, the group can better control the propagation of information in the network. This allows  $m$  to slow down the propagation of any concurrent block, giving it time to substitute its own. In the Bitcoin protocol and many others, the block that was first received takes precedence which further worsen the effect of the attack.

For every bloc,  $M$  is more likely to mine it first. It is however unlikely that  $M$  will mine all the blocks.  $m$  will therefore get a chance to mount the attack over time. Conversely, if  $m$  holds only 1% of

the network's computation power, the attack is unlikely to succeed as a few extra seconds working on a block will not cover the gap in computation power.

## 2.9 Blockchain security: limitations

Beyond attacks and vulnerabilities, the blockchain presents a number of limitation that are worth keeping in mind. They are presented below. The limitations stem from technical issues that are yet to be solved, from protocol constructions that will likely remain unchanged, and also from the politic and legal framework that surrounds the blockchain.

### 2.9.1 Bootstrap

A successful crypto-currency (or blockchain to an extent) rests on three pillars : Value of the currency, trust in the system and nodes engaging in the block creation process. The problem is, the three are heavily linked. For participants to engage in securing the network, the value of the currency (which they are rewarded with) must be sufficient and stable. For the currency to be stable, the system must be perceived as secure. For the system to be secure, participants must engage in its protection.

Because they are so clearly linked, building a new crypto currency from scratch and getting it of the ground is difficult. You must have participants willing to secure the network either by ideology as it has been the case in the early days of Bitcoin, or in the hope that the investment will be fruitful as it happens with alt-coins.

### 2.9.2 Privacy

Most blockchains guarantee pseudonymity: users hide behind pseudonyms (blockchain addresses). The link between a user and their pseudonym is not trivial. But each pseudonym can be linked to its activity as everything is public, including the transactions, their emitters and beneficiaries. A number of *manual* workaround exist. For example, several transactions can be bundled into one by assembling all their inputs and outputs in order to mask who is paying who.

Another solution is to owned multiple addresses. This way, transactions made from each cannot be linked. This has the added advantage of increasing security as loosing access to one address is less damaging if it does not contain all our funds. Some people even advocate to use each address only once to break usage patterns that could lead to de-pseudonymisation, such as often sending transactions to the same addresses, or invoking the same smart contract.

Reid et al. [Reid and Harrigan, 2013] have shown that pseudonymity in the Bitcoin network can be lifted by observing transactions and crossing that information with external sources. In consequence, users should be wary of what they publish on the blockchain. Also, pseudonymity is not to be confused with anonymity where no link can be drawn between operations made by a single user.

ZeroCoin [Miers et al., 2013] uses zero-knowledge to convert bitcoins into anonymous coins called Zerocoin. Zero-knowledge is a class of proofs by which no information can be gained except for the truth of the object of the proof. Coins are first committed through a mint transaction. Ownership of the coins can then be proven in a zero-knowledge fashion, keeping private the value of each coin and the address

of their owner. Imagine a blackboard to which people stick bills. When they want to pay for something, they must prove that they have stuck a bill up. They then can take whatever bill they want and pay for their purchase. That way, the used bill cannot be traced to the person that stuck it on the board, nor can it be linked to the person that unstuck it. It is simply coming from the blackboard.

Dash [Duffield and Diaz, 2015] is another example of privacy-focused coin.

### 2.9.3 Propagation delay

As we have seen, the most famous attack on the Bitcoin blockchain requires the attacker to hold more than 50% of the network computing power. Decker et al [Decker and Wattenhofer, 2013] show that this hypothesis might be a little optimistic. The article takes a closer look at the propagation of information on the Bitcoin network and reveals that blocks can take longer than 40 seconds to reach the edge-nodes, 10 minutes being the average time between two blocks. This propagation delay is responsible for a higher number of forks, which threaten the integrity of the blockchain.

Bitcoin forms a random graph, the topology of which cannot be controlled since nodes constantly leave and enter the network. Some nodes might therefore be really hard to reach. For that reason, they receive blocks after everyone else. In the meantime, they keep trying to solve the puzzle. This increases the chance that they will find a viable candidate, thus causing a fork.

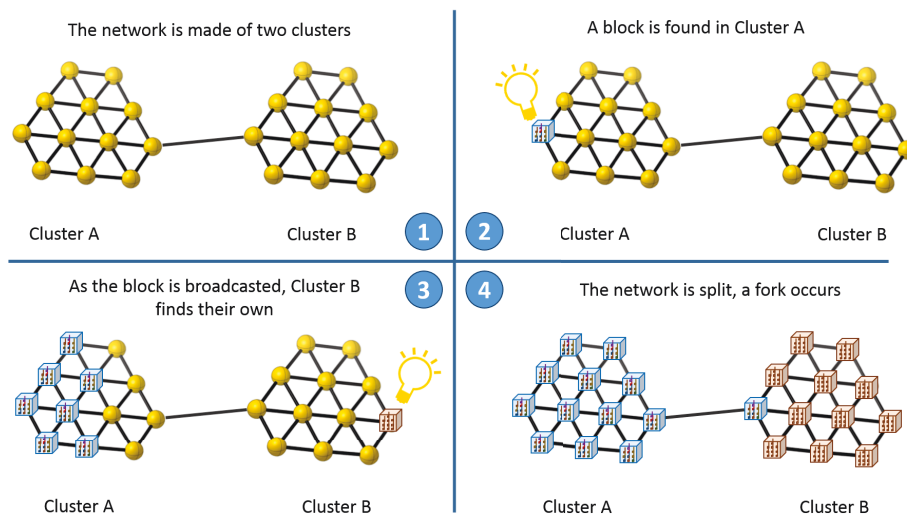


Figure 2.15: Propagation delay leading to a fork

Imagine the following pathological situation illustrated in Figure 2.15: the network is made up of two disjoint clusters of nodes bridged by a single edge. When a block is found in cluster *A*, it has to reach that one special node in order to be broadcasted in cluster *B* and then reach all the nodes in that cluster. Because of the delay, when it finally arrives, cluster *B* has produced a block of its own. Hence a fork. For the next block, the situation is similar. Once again, each cluster finds its own block.

Each block has been found thanks to half of the network computing power. If an attacker were to join cluster *A*, the power they would need to take over the consensus process would be half of cluster *A* computing power, which is half of the network's power, so only 25%. As long as the fork stands, 50% of the network is wasting power: Only one chain can survive.

Of course this topology is not realistic. Forks still affect the network to a lesser degree while delays in block propagation increase the likelihood of a fork. According to the authors, at the time of publication 49,1% of the computing power of the network would have been enough to mount a successful attack. The increase in Bitcoin and blockchain popularity is likely to further worsen this problem as it will increase the size of the graph and with it the propagation delay.

#### 2.9.4 Transaction rate

One major limitation of blockchain adoption is its throughput. Bitcoin averages around 4 transactions per second<sup>32</sup>. Ethereum can validate 25 transactions per second. The visa network validates 45 000 transactions per second. The contrast is stark.

#### 2.9.5 Legislation

Bitcoin is not own by any country. This does not mean that Bitcoin is free from nations' influence. In February 2019, Datalight published an infographic revealing the distribution of Bitcoin nodes around the world (see Figure 2.16). We learn that ten countries host just under three quarter of the nodes, while the top three (the United States, Germany, and France) account for over 50% of the nodes (5339 out of 10579).

Similarly, the mining pool landscape is dominated by China. Decisions from the Chinese government have had drastic effect on Bitcoin's operation in the past<sup>34</sup>. The censorship of Bitcoin conferences or workshop in the country as well as the ban of Bitcoin-related accounts by a handful of Chinese banks led to a market crash.

At the moment, blockchains evolve in a politic and legislative fog. The concept is only ten years old. Mainstream interest is even more recent and we are far from mainstream adoption. Evolution in the matter is complicated to predict. Legislation could shape the future of blockchains, their use, and adoption.

**Legal framework** Blockchains are in their infancy. It is fair to assume that they will evolve a lot in the coming years. Early legislation could hinder the development of the technology. Countries could also end up with legal definitions that do not match realistic applications and are barely usable. This explains the void around the topic in many countries.

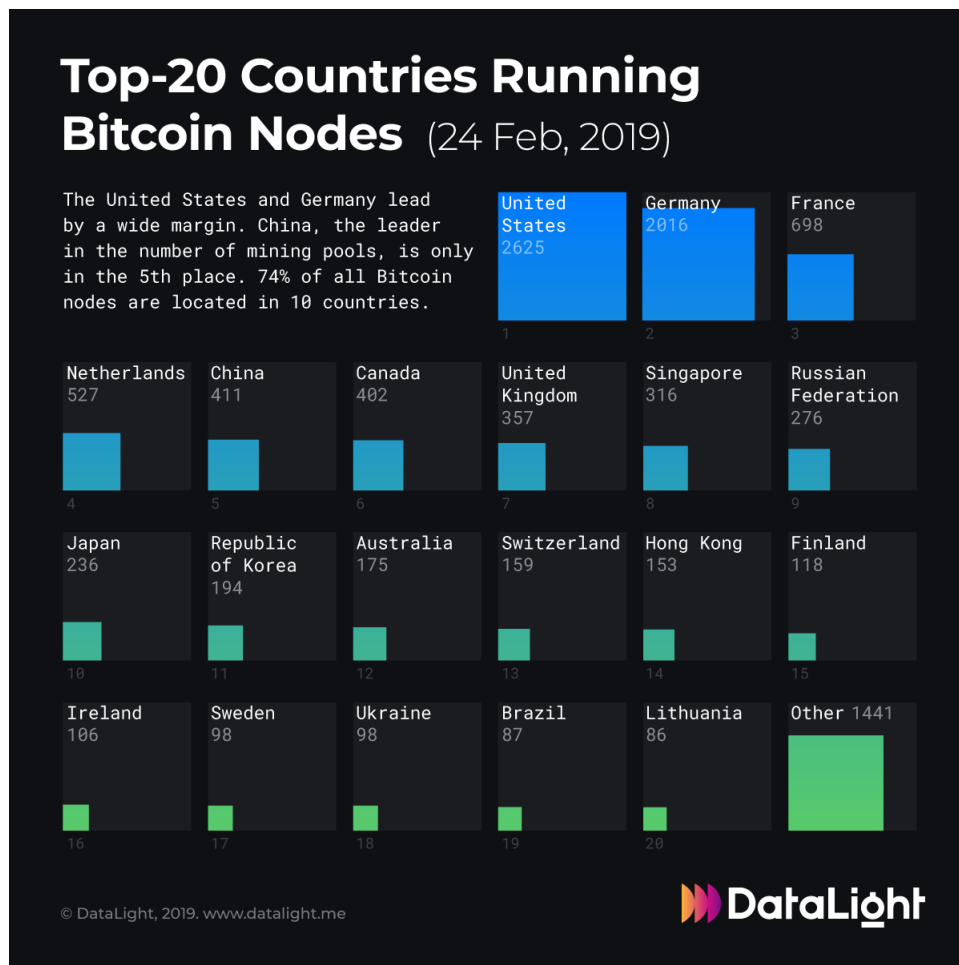
A few countries including France and the United States have started to recognize and tax cryptocurrencies<sup>35</sup>. Some others are trying it out. Sweden for instance, has launched a blockchain-based land

---

<sup>32</sup><https://www.blockchain.com/en/charts/transactions-per-second>, Last checked: July 8th, 2019

<sup>34</sup><https://www.ccn.com/china-now-controls-bitcoin-thats-just-beginning>, Last checked: July 8th, 2019

<sup>35</sup><https://www.thebalance.com/how-bitcoins-are-taxed-3192871>, Last checked: July 8th, 2019

Figure 2.16: Bitcoin Nodes around the world<sup>33</sup>

registry project that is in its demonstration phase<sup>36</sup>.

Despite these first steps, blockchain applications evolve in an uncertain legal framework. They can also be in conflict with existing laws. The General Data Protection Regulation (GDPR) has recently become enforceable (since May 2018). It is designed to protect the rights of European citizen with regards to data collection. In particular, it contains a provision that enables citizens to request their private data be erased. But by design, transactions on the blockchain are immutable. Another requirement of the GDPR is for European data to remain in Europe. That can cause another headache for public blockchain applications where the location of nodes are unknown and unrestricted.

<sup>36</sup><https://www.coindesk.com/sweden-demos-live-land-registry-transaction-on-a-blockchain>, Last checked: July 8th, 2019

Developing blockchain applications under GDPR is shaping up to be quite tricky<sup>37</sup>. Research efforts will likely concentrate on zero knowledge proof, homomorphic encryption and other privacy preserving mechanisms that could be added to the blockchain. As we have seen in Section 2.7, their integration to existing blockchains will not be painless.

**User Protection** On the other side of the coin, few protections exist for blockchain users. The blockchain community wants above all to prevent regulators from getting involved. This was part of the argument for the DAO's hard fork: the disaster was big enough and people lost enough money that regulators might want to intervene. So the community reversed the hack.

But people lose their keys, fall for scams, or are abused in other ways all the time on a smaller scale with no recourse. It is the cost of autonomy: nobody will catch you when you fall. What to do if a bug in a smart contract ends up overcharging me? The answer lays outside of the blockchain. Including real life contracts into smart contract can be a way to make them legally binding.

Until blockchains have matured and laws are put in place though, it seems like users and developers alike will have to navigate in the dark.

## 2.10 Blockchain & IoT

We have presented the blockchain, plunged into its inner workings, taken a look at some implementations, and analyzed its security. Armed with this knowledge, we can now discuss the blockchain compatibility with IoT use cases.

Christidis et al. [Christidis and Devetsikiotis, 2016] and Fernandez et al. [Fernández-Caramés and Fraga-Lamas, 2018] have specifically studied this question in their respective papers. The present section summarizes their conclusions and adds our own.

There are several arguments for mixing blockchain and IoT:

- **Decentralization** - One of IoT's current challenges is the departure from centralized models and a migration toward edge intelligence where gateways and devices are given a bigger role, reducing dependencies to cloud servers. The blockchain offers resilience, availability, resistance from tampering, and all other advantages that come with decentralization.
- **Desintermediation** - Blockchains do away with our reliance on trusted third parties. They allow users and developers to evolve on a platform that belongs to them and define the terms of their interactions themselves. IoT devices can interact directly with one another in a secure manner. Users can create and manage their own identities and regain some ownership over their data.
- **Transparency** - There is a demand from users for more transparency. They want to know what is being done with their data. The blockchain also enables the tracking of IoT devices' activity, thus ensuring correctness.

---

<sup>37</sup><https://medium.com/wearetheledger/the-blockchain-gdpr-paradox-fc51e663d047>, Last checked: July 8th, 2019



- **Reduced operation cost** - The blockchain offers a shared architecture with built-in security and, for some, a billing service. The lack of intermediary decreases operation costs. Lengthy workflow can be verifiably automated.
- **Auditability** - In the event of an attack, blockchain data are available for a forensic analysis.

### 2.10.1 Use Cases

The advantages offered by the blockchain do not come free. In particular, there are performance limitations that one must consider before deploying a blockchain-based application. Fernandez et al. [Fernández-Caramés and Fraga-Lamas, 2018] provides a useful decision tree to determine whether a blockchain is a good choice for a given application. We reproduce it as Figure 2.17.

The following use cases illustrate the benefits we have identified above.

**Use Case 1 - Software Update** Let us consider  $n$  car manufacturers that regularly rolls out updates for their products. Currently, the manufacturer has to maintain a server where the software can be downloaded. If someone maliciously takes over the server, there is no way for devices or users to know that the file they are downloading is incorrect until the breach is revealed.

Using the blockchain, the manufacturer puts up a notice for the latest software update and embeds its hash in the transaction. Clients can then download the file and compare its hash to the value stored in the blockchain. Cars can also request the update from each other, waiting for the light to turn green. Now the vendor only needs to get the software update to enough clients in the beginning and let them take care of the rest. If the manufacturer goes out of business or if its server is taken down, clients can still get the updates in a secure manner.

**Use Case 2 - Supply Chain** Let us consider a clothes company with many subcontractors. Currently, the company trust their subcontractors to be responsible for the product they order and have no visibility on the potential subcontractors they themselves hire. There is a demand for more transparency from the clients, in particular with regards to the conditions the clothes were made into.

The blockchain can be used to track the product's journey. Every actor in the chain is given a blockchain address and a device to automatically register the change in ownership. Trucks are given the same. When devices from two different actors and the truck are in range, they sign a blockchain transaction transferring ownership of the truck and its content to the next person in the chain. That transaction is only valid if signed by the three party. It can also contain the current state of the shipment, or any other relevant information.

**Use Case 3 - Electricity Market Place** Let us consider individuals with solar panels. Their panels produce more energy than they consume. Currently, individuals must sell this energy to the electricity company that then resells it to other costumers.

Using the blockchain, individuals could sell directly to other individuals and pay a portion of the profit to the electricity company for the use of their delivery network. There is no need to implement a

payment system. Individuals advertising false bids can be sanctioned. The cost of entry is minimal for the participants.

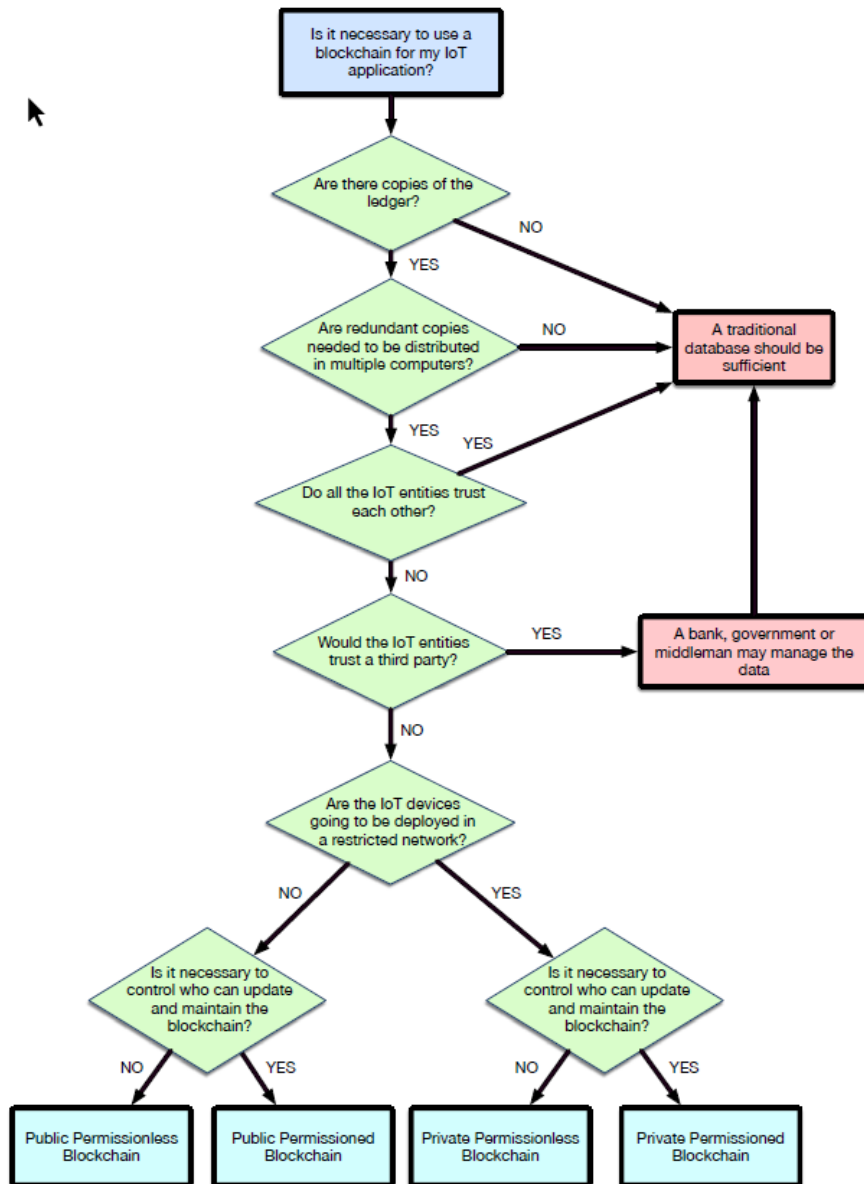


Figure 2.17: Should you use a blockchain? Taken from [Fernández-Caramés and Fraga-Lamas, 2018]

### 2.10.2 Limitations

If blockchains and IoT seem like a good match, let us take a step back and present some counter arguments to this marriage. The following topics require some work from the community:

- **Scalability** - As we have seen in Section 2.9.3, the propagation delay induced by the network topology is a security risk. More nodes will only worsen the problem. Additionally, the transaction rate is limited which is a real hindrance to scalability.
- **Transaction validation time** - The blockchain offers no guarantees in terms of transaction validation time. Time sensitive use cases should not be considered.
- **Resource Efficiency** - IoT devices cannot host blockchain nodes at the moment. The bandwidth requirements, the storage space required, the type of cryptography that is used, ... All of these are not IoT friendly. A delegated or closely related gateway can be used however to mitigate this.
- **Stability** - Applications need a stable cryptocurrency in order to actually use the blockchain as a billing system. The price volatility also complicates the cost estimation for vendors. A more stable ecosystem would be preferable. Standardization would be a great help in the matter. It would also help with adoption as actors are currently highly fragmented.
- **Privacy** - As we have seen in Section 2.9.2, privacy can be a real concern on the blockchain.
- **Usability** - Blockchain application are not always the most user friendly. A discovery service or a DNS to make blockchain addresses humanly readable would be an improvement.
- **Test environment** - Developers need a stable testing environment. Not all blockchains provide that today. Ethereum for instance has a test network where smart contract can be tester before being deployed and mining is easy enough for tests to be cheap to run.
- **Censorship** - Miners have a huge impact on a blockchain. They can choose to censor transactions. Special attention should be taken when choosing miners.

## 2.11 Conclusion

In order to efficiently use a tool, one must first understand it. In this chapter, we have presented the fundamentals of the blockchain technology and its complicated governance model. We have reviewed security issues and potential limitations. Based on this information, we have discussed the relevance of the blockchain for IoT applications.

The blockchain is still an emerging technology. As such, it comes with a host of issues that may or may not be resolved in the coming years. However, the blockchain community has shown that solutions can be devised and problems circumvented. Amongst these issues, an IoT-optimized blockchain that would enable devices to connect to the blockchain without the need of a server or gateway is yet to emerge. Despite this, the blockchain offers a host of properties that can be interesting for IoT applications.



# Chapter 3

## Survey : Access control in the IoT

### Contents

---

<b>3.1</b>	<b>Introduction</b> . . . . .	<b>64</b>
<b>3.2</b>	<b>Background</b> . . . . .	<b>66</b>
3.2.1	Access control glossary . . . . .	66
3.2.1.1	Abstraction levels . . . . .	66
3.2.1.2	Formal model . . . . .	68
3.2.1.3	Actors . . . . .	68
3.2.1.4	Delegation . . . . .	69
3.2.1.5	Architectural Element . . . . .	69
3.2.2	Capability-based access control (CapBAC) . . . . .	70
3.2.2.1	Concepts . . . . .	71
3.2.2.2	Token format: self-descriptive vs opaque . . . . .	72
3.2.3	Attribute-based access control (ABAC) . . . . .	72
3.2.3.1	Concept . . . . .	73
3.2.3.2	ABAC vs RBAC . . . . .	73
<b>3.3</b>	<b>Comparison criteria</b> . . . . .	<b>73</b>
<b>3.4</b>	<b>Centralized architecture</b> . . . . .	<b>76</b>
3.4.1	No serverless authorization . . . . .	78
3.4.2	Serverless authorization . . . . .	79
3.4.3	Conclusion . . . . .	81
<b>3.5</b>	<b>Hierarchical Architecture</b> . . . . .	<b>82</b>
<b>3.6</b>	<b>Federated Architecture</b> . . . . .	<b>85</b>
<b>3.7</b>	<b>Distributed Architecture</b> . . . . .	<b>90</b>
3.7.1	PDP/PEP hybrid . . . . .	90
3.7.2	Multi PDP . . . . .	94
3.7.3	Blockchain-based solutions . . . . .	96

3.7.4	Conclusion . . . . .	100
<b>3.8</b>	<b>Taxonomy &amp; Analysis . . . . .</b>	<b>101</b>
3.8.1	Analysis . . . . .	101
3.8.2	Taxonomy . . . . .	101
3.8.3	Future research direction . . . . .	102
<b>3.9</b>	<b>Conclusion . . . . .</b>	<b>104</b>

---

## 3.1 Introduction

The goal of this chapter is to present and analyze access control solutions for IoT applications proposed in the literature. It is based on a survey currently under review by the journal ACM Computing Surveys<sup>38</sup>.

**Contributions** The contributions of this chapter can be summarized as follows:

- Definition of classical access control architectures based on the four core functions of access control (PEP, PDP, PAP, PIP)
- Highlight of each reference architecture’s strengths and shortcomings
- In-depth review of the literature organized by architecture
- Highlight of each reference’s strengths and weaknesses
- Evaluation of solutions based on both objective and comparative criteria
- Guide for the reader to choose an architecture adapted to their needs
- An architecture-oriented taxonomy of access control solutions
- Potential future research direction for access control in the IoT

**Related Surveys** IoT security has been a subject of interest in the academic community for around a decade. There are therefore a few surveys covering the topic. If some papers [Sain et al., 2017, Ouaddah et al., 2015] present a short summary of trends and challenges faced when devising access control solutions for the IoT, the majority does not focus on access control.

Roman et al [Roman et al., 2013], Sicari et al. [Sicari et al., 2015], and Tourani et al. [Tourani et al., 2016] cover some of the main security issues facing the research community, amongst which authentication, confidentiality, privacy, trust, policy enforcement, and access control. Yang et al. [Yang et al., 2017] analyze IoT security from the perspective of the different layers: perception, network, transport, and application. Here access control is mentioned once again without being focused on. Ammar et al. [Ammar et al., 2018] are interested in the overall security of commonly used frameworks such as AWS, Azure,

---

<sup>38</sup>Originally submitted in February 2018, resubmitted after a major revision request in March 2019

or ARM mbed IoT. It tackles their hardware and software dependencies, their use of industry standards, their handling of device-to-cloud and cloud-to-user communication, their security-related functionality, and their efficiency at protecting user's data which involves access control. Khan et al. [Khan and Salah, 2018] categorize IoT threats and challenges according to their severity and puts forwards states of the art solutions for each identified issue. A few access control solutions are mentioned as a means of mitigating common attacks such as DDoS and achieving end-to-end security.

Kouicem et al. [Kouicem et al., 2018] are interested in examining the impact of new emerging technologies that are the blockchain and Software Defined Networking (SDN) on security solutions in contrast with more classical cryptography-based techniques. Access control is presented as a security mechanism among others needed to ensure availability.

Access control is the specific subject of Zhang et al. [Zhang et al., 2018a] but the paper's scope is limited to fog computing. Ouaddah et al. [Ouaddah et al., 2017b] presented the first in-depth survey dedicated to access control solutions for the IoT. It structures its analysis around the OM-AM (Object, Model, Architecture, Mechanism) authorization reference model [Sandhu, 2000]. Consequently, solutions are separated according to what model or protocol they implement: Role-Based Access Control (RBAC [Ferraiolo and Kuhn, 1992]), Attribute-Based Access Control (ABAC [Hu et al., 2013]), Usage Control (UCON [Park and Sandhu, 2004]), Capability-Based Access Control (CapBAC), OAuth [Hardt, 2012], UMA [Maler et al., 2015], etc.

Bertin et al. [Bertin et al., 2019] take a similar approach. They articulate their reflexion around two axis: access control abstractions (cf. section 3.2.1.1) - Discretionary AC, Mandatory AC, RBAC, ABAC, etc - and standards - SAML, OAuth, ACE, UMA, etc.

**Architectures** Our proposed survey is focused on the architecture instead. The same choice was made in 2013 by Roman et al. [Roman et al., 2013] for the generic topic of IoT security. However, a lot of work has been published since. We also use different definitions for our architectures that are based around the different sub-functions of access control.

We articulate our analysis around 4 classical architectures - centralized, hierarchical, federated, and distributed.

Our architectures are built on the core most resource demanding function: the access control decision (cf. section 3.2.1.5) which requires memory to store policies, computational power to run, and, optionally, bandwidth to retrieve information required for the decision or to communicate said decision to an enforcer. The localization of this function and its multiplicity therefore has a strong impact on the efficiency and capacities of the system.

Analyzing existing solutions through that prism is advantageous for the following reasons. First, it allows us to identify architecture-specific limitations or advantages that then transfer to standards and protocols that adopts them. Second, it presents operators or integrators with a way of selecting access control solutions suited for their own infrastructural needs regardless of the standards they implement. Third, we contribute to the ongoing discussion of decentralization in the IoT by providing a precise definition of different architectures, including hierarchical and federated, that are often overlooked, an in-depth analysis of their theoretical benefits and disadvantages, and an overview of papers that implement them.

**Organization** The rest of this chapter is organized as follows. Section 3.2 introduces terms and notions used throughout this thesis. Section 3.3 discusses the criteria used to compare access control solutions. Section 3.4 presents centralized solutions from the literature. Section 3.5 reviews proposals with a hierarchical architecture. Section 3.6 looks at federated architectures. Section 3.7 proposes three definitions for distributed architectures. Section 3.8 provides an overall analysis of the previous sections, details our taxonomy, and discusses future research directions. Finally, Section 3.9 concludes this chapter.

## 3.2 Background

The background presented below is relevant not only to this chapter but to the thesis as a whole. Additionally to fixing the definition of frequent terms, it takes a closer look at CapBAC and ABAC, that are the basis of solutions presented in Chapter 4 and 5 respectively.

### 3.2.1 Access control glossary

In this section, we put forward precise definitions for the access control-related vocabulary used throughout this thesis. We present the different abstraction levels involved in access control systems and illustrate their differences with classic examples. We detail vocabulary used in formal model, the different actors, delegation terminology, and the architectural blocks that make up access control systems. These blocks will be used to define the architectures presented in Sections 3.4 to 3.7.

When no sources are cited, definitions are taken from the IETF Internet Security Glossary [Shirey, 2007].

#### 3.2.1.1 Abstraction levels

Access control systems can be divided into three abstraction levels that correspond to different phases of development. From the most abstract to the most concrete: policies, models, and mechanisms. Considering each level independently leads to a greater freedom in designing the system: security proofs can be done separately and according to each level's requirements, different models can be used to represent the same set of policies, mechanisms can be changed throughout the system's life, etc. That last point is especially important for a system's longevity and adaptability. If some mechanisms are constructed with specific models in mind or seem more adapted to others, the most popular policies, models, and mechanisms are orthogonal to one another. Finally, real-life systems will often mix several solutions to better fit their requirements.

**Access control policy** An access control policy is defined [Shirey, 2007] as a plan or course of action that is stated for a system or organization and is intended to affect and direct the decisions and deeds of that entity's components or members. Access control *policies* are therefore most often defined by organizations or applications and pertain to who should access what, based on logics that are usage-dependent. They define the (high-level) rules according to which access control must be regulated [Samarati and de Vimercati, 2000]. Examples of well-known access control *policies* include Discretionary Access Control (DAC [Qiu et al., 1985]) and Mandatory Access Control (MAC [Qiu et al., 1985]).



In **DAC**, resources belong to their creators that have full discretion over the permissions that should be associated to them. This type of policy is for example used in classical file systems. We take the example of a medical file in a hospital. Under DAC, the doctor that originally wrote the file owns it and fixes its access conditions. DAC is highly flexible and user-centric. However, no safeguards are in place to protect owners against mistakes they might make and administration can get tricky. This is particularly true when permissions need to be revoked. When a doctor leaves the hospital for instance, they should have loose access to a patient's medical information.

**MAC** on the other hand imposes centrally-defined policies on all resources in the system. The historical use case for MAC is the military sector. Taking our medical file example, under MAC the hospital decides who can access and modify each file. MAC is better suited to organizations. However, it is really rigid and requires rules to be set centrally and in advance for all resources that may be considered in the system.

**Access control model** Access control models [Samarati and de Vimercati, 2000] provide a formal representation of the access control policy and its workings. This formal representation provides the framework to prove security properties of the designed access control system. *Policies* can evolve over time to adapt to new requirements or changes in organization. *Models* should support these changes. Examples of well-known access control *models* include RBAC [Ferraiolo and Kuhn, 1992] and ABAC [Hu et al., 2013].

In **RBAC**, a subject's permissions depend on the role it is affected to. The hospital's staff would likely be assigned roles corresponding to their function (doctor, nurse, janitor). A subject is then allowed to access medical files if it is a doctor. Grouping subjects increases manageability. It is also a natural way of translating roles within an organization/company into the system. Consequently, the burden of security is put on role definition, a tricky process that represents a big entry cost. For that reason, roles are generally defined to be static. When fine-grained permissions are required, roles need to be more fine-tuned. This can lead to an over-complication and role proliferation.

**ABAC** takes authorization decisions based on attributes of subjects, objects, actions, or environment. A doctor is given an attribute *IDENTITY* of value *john\_dorian* and a *PROFESSION* attribute of value *doctor*. The file is given two attributes *DATA\_CLASS:medical* and *REFERRING\_PHYSICIAN:john\_dorian* so that one must not only be a doctor to modify it, it must also be the patient's own referring physician. As with RBAC, the definition of access rules represent a high set up cost for any ABAC system. Yet, attributes enable more flexibility as they can be applied not only to subjects but to objects, environment variables, and actions. Entities can have several concurrent attributes that can be issued or revoked over time, leading to more granularity. This may however require complex rules and heavy computations. Conflicts may also arise when two rules come to a different access control decision. The resolution of such situations adds another layer of complexity to the system design. ABAC is explored in more details in Section 3.2.3.

**Access control mechanism** Access control mechanisms [Samarati and de Vimercati, 2000] define the low level (software and hardware) functions that implement the controls imposed by the policy and formally stated in the model. Examples of well-known access control *mechanisms* include Access Control Lists (ACL [Van Tilborg, 2014]) and capabilities [Dennis and Van Horn, 1966].

**ACL** associate subjects, objects, and permissions in a static manner. Each subject/resource pair must have an associated entry. The lists are organized per object. In practice, ACL can be defined using groups of either subjects (roles) or objects. Using ACL to authorize Dr Dorian to modify the medical file of a Patrick Jones would look like this: (*DrDorian,patrick\_jones.doc,write*). If M. Jones changes doctor, a new ACL is required. They are easy to implement and use. The revocation of a single permission or the removal of an object from the system are simple operations. Granularity can also be easily introduced. ACL however do not scale well. Nor are they adapted to systems with few objects and many subjects. As ACL are made with a *per object* mindset, it is difficult to view or manage the permissions associated to a subject.

**Capabilities** are tokens that carry objects and associated permissions. A single capability can hold multiple privileges. Contrary to ACL that are stored within the access control system, capabilities are given to subjects that must present them with their requests. In our case, Dr. Dorian would be given a token containing the name of the file and the *write* permission. This token would be required for each modification. Capabilities are explored in details in Section 3.2.2.

### 3.2.1.2 Formal model

An **object** is a system component that contains or receives information. We extend this definition to include all entities that can be accessed or acted upon.

A **subject** is a system entity that causes information to flow among objects or affect changes to the system state. A *subject* may itself be an *object* relative to some other *subject*; thus, the set of *subjects* in a system is a subset of the set of *objects*. When the subject is a human, the term *user* is sometimes preferred to *subject* throughout this document.

An **operation** [Hu et al., 2006] is an active process invoked by a *subject*. In the reminder of the paper, the term *action* is sometimes used to refer to an *operation*.

An **authorization** is an approval that is granted to a subject to access an object. The words *authorization*, *permission* and *privilege* are somewhat equivalent and interchangeable. Depending on the context, one or the other might be preferred. *Privilege*, for instance, should be preferred in the context of computer operating systems. In the reminder of this document, *authorization*, *permission* and *privilege* are used interchangeably.

### 3.2.1.3 Actors

The **owner** of an object is the person or organization that has the final statutory and operational authority over said object and the information it contains.

The **issuer** of a capability (see Section 3.2.2) is a system entity emitting that capacity. In order for the capability to be valid, the issuer should have the authority required for authorization operations.

An **administrator** is a person that is responsible for configuring, maintaining, and managing the system in a correct manner for optimizing security.

A **security domain** is an environment or a context that:

- (a) includes a set of system resources and a set of system subjects that have the right to access the resources.

(b) is usually defined by a security policy, a security model, or a security architecture.

A "controlled interface" or "guard" is required to transfer information between domains that operate under different security policies.

#### 3.2.1.4 Delegation

**Delegation** is the process by which a *subject* transfers all or parts of its *privileges* to another *subject*.

The **delegator** is the *subject* which delegates its *privileges*.

The **delegatee** is the *subject* that receives new *privileges* through the delegation process.

#### 3.2.1.5 Architectural Element

All access control solutions perform the same function: guard the system against unauthorized access. They therefore all use the same fundamental building blocks. These blocks represent logical functions. In practice, they can be hosted by the same system entity, such as a central server that perform access control on its own, or they can be divided amongst several collaborating actors. Their logical separation makes for an easily decentralized model. The way they are distributed amongst actors and the number of times they are replicated is what we will use to define an architecture.

We refer to these blocks sometimes as architectural elements, sometimes as logical functions, and sometimes as points, for system endpoint. The four elements and their interactions are depicted in Figure 3.1.

**Policy Decision Point (PDP)** [Yavatkar et al., 2000] The point where policy decisions are made.

**Policy Enforcement Point (PEP)** [Yavatkar et al., 2000] The point where the policy decisions are actually enforced.

**Policy Information Point (PIP)** [Hu et al., 2013] An entity that serves as the retrieval source of attributes, or the data required for policy evaluation to provide the information needed by the PDP to make the decisions. A PDP can query a number of different PIP to retrieve contextual information on the subject, the object, the environment, or other components of the system, allowing for more complex access control policies.

**Policy Administration Point (PAP)** [Westerinen et al., 2001] The system entity that creates a policy or policy set.

Let us illustrate the interactions depicted in Figure 3.1 with a simple policy example: for all subject  $s$ , if  $s$  is an employee, then  $s$  can open the parking door.

First, the PAP defines this rule and pushes it to the PDP. Assume it is Monday morning and Belinda is coming to work. She presses her badge to the parking door, the PEP. The PEP first retrieves her identity from her badge. It then interrogates a server somewhere in the building that acts as a PDP. The PDP

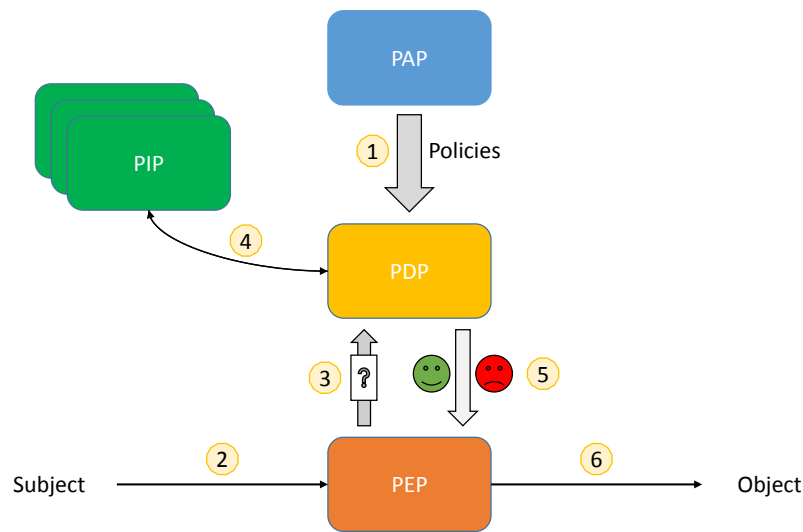


Figure 3.1: Instance of access control request

queries a database acting as a PIP to ask whether Belinda is an employee. The PIP confirms that Belinda is indeed an employee. A positive response is sent back to the PEP that opens the door for her.

In the above example, Belinda’s badge is a proof of her identity. In reality, her badge could be used just as easily to prove her employee status as it was delivered by her employer. It could also be used to directly store access rights to the parking door, her office, the cafeteria. The PEP would simply need to validate that the permissions were issued by the employer. Her badge would then act as a capability.

### 3.2.2 Capability-based access control (CapBAC)

As the rest of this chapter will prove, the use of capabilities (often referred to as tokens) is really popular in the IoT access control literature. Tokens are indeed well suited for IoT uses cases as they enable a clear separation of the PEP and PDP functions. Additionally, they impose very few communication patterns between entities and are compatible with serverless authorization. Our contribution on the topic of token libraries is detailed in Chapter 4.

In this section, we present the concept of Capability-Based Access Control and discuss different token formats.

### 3.2.2.1 Concepts

The concept of capability was first introduced by Dennis and Van Horn in 1966 [Dennis and Van Horn, 1966]. It is defined as “a token, ticket, or key that gives the possessor permission to access an entity or object in a computer system”.

A more formal definition [Van Tilborg, 2014] is as follows:

**Definition 3.2.1** (Capability). *An individual capability is a pair  $(o, (r_1, \dots, r_n))$ , where  $o$  is the object and the  $r_i$  are access rights for  $o$ .*

Capabilities materialize an authorization. They are therefore issued by the PDP, and presented to the PEP. In CapBAC, a capability must be presented with every access request. A request that does not carry a capability or that carries a capability with inappropriate permissions is rejected. This guarantees complete mediation [Saltzer and Schroeder, 1975], a property that requires every access request to be checked for authority. This implies any request must be intercepted by the PEP, and authorized by the PDP. The two events can happen in any order. Everyday-life is full of capabilities, as illustrated by the two following situations.

We go back to Belinda that wants to enter a secure area at work. She holds a badge containing the required permissions. Her photo is also on the badge. She scans it to the door. A security guard checks if the picture corresponds. The badge is the capability. It must be presented with each access request, contain the right permissions, and the right photo. Otherwise, access is denied. The photo is a reference to physical characteristics that are unique to Belinda and can be used to identify her, linking her to the capability.

Now the same Belinda wants to go backstage at a concert. Another security guard stands in her way. She presents a VIP pass and is granted access. Here, the VIP pass is the capability. It does not contain a reference to the subject. If the pass is stolen, the permissions it represents are transferred to the one that stole it, which, in this case, is considered acceptable.

This touches on the first of a few properties of capabilities:

**Property 3.2.1.** *Capabilities can require proof of ownership to be used.*

Note that Definition 3.2.1 makes no reference to a subject. A capability needs not reference the subject for which it was issued. In this case, privileges contained within are transferred to any subject that holds the capability. Depending on the situation however, a link between capability and subject might be desirable. Digital signatures can be used to create such a link. When no link exists, ciphering of the communication channels, or storage in a smart card mitigates the risks of theft.

**Property 3.2.2.** *Capability-Based Access Control has built-in support for delegation.*

A capability can simply be transferred to delegate the rights within. The cost of this is that it is hard to determine what subjects can access an object at any given time. It is complicated to get a per-object view of the system. This causes the revocation of permissions to be a cumbersome process in the case of serverless authorization.

**Property 3.2.3.** *Capability-Based Access Control supports serverless authorization.*

Capabilities can be used in such a way that after they are issued, no contact with the issuing authority is needed to verify them. It is the case for both the VIP pass and the company badge.

### 3.2.2.2 Token format: self-descriptive vs opaque

We can distinguish between two types of capabilities: **self-descriptive** and **opaque**. Choosing one or the other impacts the mechanisms for issuing and using capabilities. The former offers support for serverless authorization (Property 3.2.3) while the latter does not. Frameworks that use tokens such as OAuth 2.0 [Hardt, 2012] do not usually specify a format for the token.

A **self-descriptive capability** is usually a signed string containing information such as the object, the subject of the authorization, a set of operations, and the issuer of the capability. The token might include any additional information pertaining to the access request. When presented with the capability, the PEP needs to be able to check its authenticity. This might include checking the issuer's signature, and even performing some access control decision based on the conditions included in the token (see Section 3.7). If the subject is mentioned, proof of identity might be required such as providing a valid signature corresponding to the public key information included in the token.

Some standard formats exist for such tokens. JSON Web Tokens (JWT [Jones et al., 2015b]) in particular are very popular, often used as a payload for either JWS (JSON Web Signature [Jones et al., 2015a]) or JWE (JSON Web Encryption [Jones and Hildebrand, 2015]) tokens. If the token is simply signed, then it carries information that can compromise the privacy of the users. If it is encrypted, a way must be devised for the user to recover the key used for the encryption.

Revocation of self-descriptive tokens can be cumbersome. It might require maintaining a list of all revoked capabilities. Updating this list can either be accomplished by the issuer pushing new revocation notices or by regularly synchronizing the local list.

For that reason, self-descriptive tokens will often include a validity period. This period results from a compromise between security and usability. If a capability is stolen, the time frame during which the thief can use it is limited. In exchange, legitimate users need to renew their capabilities regularly. But no revocation mechanism needs to be implemented. The revocation is implicit.

**Opaque tokens** need to be interpreted. The issuer chooses a random string to represent a key to the information corresponding to the capability in its system. It has the advantages of not compromising the user privacy but each use requires the issuer's interpretation of the token. This increase in bandwidth requirement may be unacceptable for some IoT applications.

Opaque tokens do not support serverless authorization. With opaque tokens however, revocation is easy. When an access request is made, the issuer is contacted to verify the content of the token. The issuer takes this opportunity to indicate that the token has been revoked. When contacted, the issuer may even challenge the original requester to prove its identity or the possession of given attributes.

In the remainder of this document, the terms *capability* and *token* will be used interchangeably.

### 3.2.3 Attribute-based access control (ABAC)

ABAC is not the most popular IoT access control model. Its flexibility and built-in decentralization however makes it compatible with federated IoT use cases. Chapter 5 presents our contribution on the subject.

### 3.2.3.1 Concept

ABAC is defined as *A logical access control methodology where authorization to perform a set of operations is determined by evaluating attributes associated with the subject, object, requested operations, and, in some cases, environment conditions against policy, rules, or relationships that describe the allowable operations for a given set of attributes.* [Hu et al., 2013]

In ABAC, two main functions need to be implemented : attribute retrieval and policy evaluation. When an access request is processed, the PDP interrogates PIP to retrieve object's and subject's attributes, along with potential environmental context. The policy is then evaluated on the bases of the retrieved attributes.

First described in 2003, XACML [Moses et al., 2005] is one of the earliest example of Attribute-Based Access Control. Earlier access control solutions tend to be based on the subject's identity. This is the case of RBAC.

### 3.2.3.2 ABAC vs RBAC

One of the disadvantage of using RBAC would be the inability to integrate multiple factors in the authorization decision: An employee will most likely be given a role that corresponds to its job within the company such as developer, human resources director, lawyer, etc. However, employees of different positions work on the same project. A subject's role is not enough in this situation to assess whether they should be allowed to access a given project's documentation.

A more granular definition of roles, such as developer on project A, will lead to role proliferation, one of RBAC's pitfall. Furthermore, roles are defined in a static manner. They are not meant to be updated regularly. Similarly, subjects are assigned to roles in a long-term fashion. New subjects therefore need to be enrolled. RBAC and other Id-based models do not fare well with dynamic users, which is an important component of some IoT use cases.

In contrast, allowing policies to be based on several subject's and object's attributes, grants ABAC more expressive power. The issuance of a new attribute to a subject does not affect other subjects, nor does it affect its earlier permissions, unless the new attribute is directly involved in a policy. This allows for more regular updates to the system, offering a greater flexibility.

In ABAC, policies can be defined without any reference to a subject nor to an object. The introduction of either is a simple operation. Rules can therefore be defined without knowing subjects or objects, and be as generic as one wishes. They can even apply to subjects from an outside organization, providing attributes can be retrieved, thus increasing interoperability.

## 3.3 Comparison criteria

Designing for the IoT represents a challenge (see Section 1.2). The most notable one is the fact that most of the nodes are constrained either memory-wise, computationally or power-wise, and can work with intermittent network connectivity. Additionally, devices might be switched off and therefore not available for long periods of time. This introduces a delay in update deployment. But most of all, the

list of clients wishing to access a device is often dynamic. Furthermore, IoT devices are used for many different use cases and different roles within each use case. This leads to a heterogeneous landscape.

The following 16 criteria try to capture the level to which each solution tackles these challenges. They can be divided into two categories: objective and comparative. The latter are used in a comparative capacity and help differentiate between solutions. The lack of easily accessible implementation prevents the objective evaluation of criteria such as resource efficiency or scalability. We instead base their evaluation on the authors description and commentary. They have no value on their own but must be considered with regards to the evaluation score of other solutions.

Additionally and when applicable, solutions will be compared on their access control model, the standards they use, specifically cryptography-wise, the use case they focus on, whether they provide an authorization or authentication solution, and where they store their policies.

The importance of the metrics presented below are application-dependent. The weight that each should be given will vary with the specifics of the use of the access control system. For smart home devices and other user-centric applications, usability might be considered a more important feature than scalability. On the other hand, the latter will be of highest importance for industrial IoT solutions.

**Registration/Bootstrap (Objective)** Before an access control solution can be deployed, secrets need to be provisioned, users registered, policies defined, etc. This phase can be considered out of scope: in token-based solution, the issuance phase is often disregarded. The complexity and usability of a solution is however heavily impacted by these initial steps. This criterion notes whether a solution describes the bootstrap process.

**Resource efficiency (Comparative)** This criterion looks at how the access control solution impacts resource consumption. Its evaluation is based on three aspects: bandwidth (number of extra messages sent and/or size of the messages), memory, and computation. This criterion is also to be considered for all actors, namely the PEP, the PDP and the subject.

**Resilience/Robustness (Comparative)** This criterion wonders how well equipped the system is against attacks such as DDoS attacks or the compromise of a device within a security domain.

**Serverless authorization (Objective)** When assessing solutions we look at how much of it can be done offline. In cases where the authorization decision is taken by a PDP outside of the device, serverless solutions do not require a connection to said PDP at access time. This enhances the autonomy, efficiency, and availability of a solution.

**Scalability (Comparative)** IoT networks are expected to connect millions of objects. Access control solutions on such system are obviously expected to scale. Scalability might concern: the number of objects, the number of subjects and their requests, or the number of access control policies. This criterion looks at how these changes affect the service and in particular its relative response time.



**Maintainability (Comparative)** Ease of physical access, network quality, a device's continuous workload, etc are aspects that can make updates and maintenance operation tricky, be they physical or remote. For this reason, the maintenance of the access control solution should affect devices and services as few as possible. Maintenance operations include deploying new keys, new certificates, associating a device with a new manager, etc.

**Permission Updates (Comparative)** A device operating offline, a lossy or busy network, can delay the application of a change in permissions. This criterion is interested in the steps needed to update permissions, and in the time separating issuance and application. This time is expressed relatively to an event such as an access request, or the issuance of a new token.

**Usability (Comparative)** Many IoT device are destined to be handled by owners that are not tech-savvy. To ensure the access control solution will be used efficiently, usability is paramount. The addition of security measures should not affect a user's experience. For instance, if several endpoints are to be contacted in order to perform an action, the user needs not know about the complexity of the underlying protocol. A user will prefer having to register once, and registration should be a straightforward workflow. In short, when possible, the steps requiring an active human intervention should be kept to a minimum.

**Granularity (Objective)** This criterion is concerned with the granularity of the authorization. Possible levels of granularity include device-level, application-level, or resource-level.

**Context-Awareness (Comparative)** This criterion is concerned with the expressiveness of access control policies, namely the capacity of an access control decision to take into account some context variables such as the subject's identity, its history, the time at which the request was placed, the security context, the network's availability, or the device's workload.

**Revocation (Objective)** Some access control solutions do not expose a revocation mechanism. It is nonetheless an important step in the life-cycle of privilege attribution. Revocation can take two forms: implicit or explicit. The former is a built-in system that does not require any intervention such as an expiration date on a token. The latter requires an active operation. A device maintaining a revocation list updated by an administrator is a good example of explicit revocation.

**Delegation (Objective)** This criterion looks at whether a delegation mechanism exists that would allow a subject to pass its privileges along to another subject.

**Auditability (Objective)** When a system has been compromised, logging information are key in discovering the source and timeline of the attack. We are interested in determining whether the system is able to log access attempts, and the failed one in particular.

**Privacy (Comparative)** This criterion looks at how much information about the system is leaked by the access control process. Are cloud-stored data encrypted or in clear ? What do policies reveal about the resources and subjects in question ?

**Governance (Objective)** An IoT system can range from a humble smart home with 5 devices to a federated network of thousands of devices. That begs the question of who is able to take decisions concerning the system and its security. We will qualify the first case a *single-head* governance as a single person is administrating the network. In the case of a shared governance, the term *multi-head* will be employed.

**Maturity level (Objective)** This criterion examines the conditions under which the solution has been test: formal verification, simulation in a lab, deployment in a controlled environment or on a real system.

### 3.4 Centralized architecture

The centralized architecture is defined by a single PDP serving the requests of multiple PEP. A central server will most often play the role of the PDP while IoT devices play the role of the PEP. For simplification, we consider that the policy definition function is centralized as well. This translates to a single PAP operated by a single administrator.

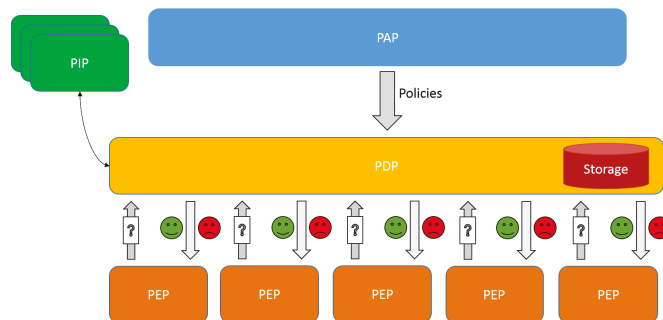


Figure 3.2: Centralized Architecture

Figure 3.2 presents the centralization of the functions of policy definition and access control decision, defined by the unicity of the PAP and PDP respectively. Here, the grant or revocation of privileges is decided by the administrator. The system contains many PEP and might contain any number of PIP. Information pertaining to policies are generally stored within the PDP.

The obvious advantage of this approach, in the context of the IoT, is to externalize a potentially memory and computationally costly process to a non-constraint environment. This enables the use of traditional and complex methods of authentication and authorization. This architecture also offers an easy administration of the system. Since the administrator is the only one authorized to affect privileges, changes only need to be enacted in one place, thus increasing usability and simplifying governance. In

addition, maintainability is increased as any change to the authorization policy only involves the central server and does not require any modification be made to devices. Since this single PDP is involved with all access requests, logs are easy to keep and audits easier to conduct.

Of course this centralization equates to a single point of failure. All information pertaining to access control is stored in one place or can be changed with one set of credentials, the loss or corruption of which would heavily compromise the security of the system. From a performance point of view, this single entity represents a bottleneck, making the system more susceptible to DDoS attacks. This, coupled with a central administration, can lead to scalability issues.

The central authority is additionally required to have access to the details of all access requests, which can be considered as compromising users' privacy. Trust in that authority is therefore required for all actors.

The central PDP approach is not compatible with policies involving device-specific contextual information, as the decision is taken outside of the device. To circumvent this issue, some solutions elect to collect and centralize contextual information from edge devices. Such queries involve a whole new set of permissions and might raise privacy issues, depending on the sensitivity of the required information, opening a new attack vector. This also implies that the device can be queried.

More concerning is the relevance the information gathered at the time the request was placed will hold by access time. The following example illustrates this issue: Let us consider a policy stating a device should only serve a maximum of 5 concurrent requests. Two subjects request access. The PDP queries the device and asks how many requests are being served. The device answers with 4, which holds true at the time. Access is granted to both subjects. Both subjects then access the device at the same time. The device is now serving 6 requests, thus violating the policy. Yet, the PDP cannot detect it, nor can it resolve the issue.

A distinction can be made between centralized solutions according to the need for the PDP to be accessible at request time. A first family of solutions, presented in Section 3.4.1, requires the PDP to be online at all time as it will be reached for a decision with every single request. Another family of solutions, reviewed in Section 3.4.2, uses capabilities or tokens to transmit the PDP's decisions to PEP. Unless this token is single-use, the subject will only need to contact the PDP again when its current token becomes invalid.

This distinction influences a solution's response to a number of criteria:

- With serverless authorization, scalability in the number of subject's request is enhanced. Fewer messages are being exchanged so bandwidth usage is lower. The system is also more usable as the user does not experience any delay that would be due to the round trip to the PDP. The messages are however bigger as they must contain the capability.
- On the other side, if the PDP is consulted before every access, revoking a permission is immediate. The access logs can be stored in the PDP that is usually not as constrained. Context parameters are evaluated at access time, allowing for the decision to be taken in a more meaningful context. For instance, an access policy calling for proximity to the device requires a real time evaluation of this parameter.

The following sections present references for each family of solutions.

### 3.4.1 No serverless authorization

A classical illustration of centralized architectures is presented by Fernandez et al. [Fernández et al., 2017]. Policies are stored outside of both the PAP and PDP, allowing for more flexibility. In a departure from the most common IoT role attribution pattern, devices are here considered as subjects. Objects are held by services, typically REST API. The Identity Provider acts as a PIP and stores the mapping between devices and roles. Policies are defined using RBAC. Each service has its own PEP. The use of a well-known protocol such as OAuth 2.0 and its Implicit Grant ensures interoperability with non-IoT systems. The burden on the device is minimal: it must implement an OAuth 2.0 client and store OAuth credentials. Access control is provided as-a-service to different applications, allowing data from IoT sensors to be posted to the cloud. Registration is required. The authentication function is performed by the Identity Provider and materialized by an opaque token. The PEP then contacts the Identity Provider to verify this token and is returned public information identifying the requesting device along with its assigned role.

This use of a token to materialize authentication is also used by Bandara et al. [Bandara et al., 2016]. Upon successful authentication, the Authentication Manager delivers a self-descriptive, digitally signed token to the user that contains the method used for authentication and its strength. The *strength* parameter varies between 0 and 1, and is introduced to specify the confidence level that can be allotted to this authentication. The most critical functions will only be accessible to users that have been strongly identified. Separating authentication and authorization into two different tokens increases usability. A user will be more likely to use two-factor authentication or other involved methods if they stay authenticated across multiple access request.

Ashibani et al. [Ashibani et al., 2017] also use confidence levels calculated using the method of authentication, contextual information such as user's profile, location or historical information. The administrator can choose which of these parameters should take precedence depending on the use case. For instance, when changing the temperature setting in a room, the user's location is more important than its profile. Contextual information are constantly being updated without the user's intervention. Permissions include a duration. Changes in context will also cause allotted permissions to be revoked. The paper [Ashibani et al., 2017] includes results from a proof of concept evaluating the performance of the different authentication methods. Combining them causes a slight overhead that is negligible when the system is accessed from the Internet. The use of contextual information mitigates the risks of unauthorized access as unusual behavior or location would raise an alert. All access attempts are logged and accounts can be locked. These results however are coming from a controlled simulation. Bandara et al.'s solution [Bandara et al., 2016] was tested in a real smart building, the Daiwa Ubiquitous Computing Research Building<sup>39</sup>, hosting over 300 devices. Tests measuring response time indicates that beyond 50 simultaneous users, the delay in query processing becomes noticeable.

Another classical approach is to encrypt data and send them to a cloud. Access is then regulated by delivering decryption keys. In order to decrypt CP-ABE (Cyphertext Policy Attribute-Based Encryption) encrypted data, a subject must be in possession of a set of attributes that are specified in the policy and delivered by an attribute authority. By looking at the policy however, one can gather information about the nature of the data. Information can then be inferred about the subjects that would request such data,

---

<sup>39</sup>[http://www.u-tokyo.ac.jp/en/whyutokyo/hongo\\_hi\\_003.html](http://www.u-tokyo.ac.jp/en/whyutokyo/hongo_hi_003.html), Last Checked: March 8th, 2019

compromising the privacy of both the data owner and data recipient. Hao et al. [Hao et al., 2019] solve this information leakage by hiding policy attributes. A fuzzy attribute positioning mechanism based on garbled bloom filters is then used to help legitimate users recover that information. This scheme is secure against dictionary attacks, meaning that even an attacker knowing all of the system attributes cannot guess which ones were used in a specific policy. In such a setting, the party that originally encrypts the data is the PDP. The decryption algorithm is the PEP as it actually enforces the policy.

Yan et al. [Yan et al., 2019] takes a very similar approach of extracting data from devices, encrypting them, and sending to the cloud. The scheme is focused on identity rather than attributes. The focus is on privacy leakage in smart home frameworks with multifunctional devices. All subjects are considered to be mobile applications. Traditionally, policies link device-wide permissions to the phone rather than the application. That can be abused when the original app is compromised or by other malicious apps running on the same phone. In this scheme, the gateway is both the PDP and the PEP. When a user first installs an app, they set the permissions for the application on a per function basis instead of per device. The gateway provides the application with a decryption key for each allowed function. At access time, a challenge message is generated and encrypted. If the subject successfully decrypts it, data is retrieved from the cloud or the request is passed along to the device. The paper lacks revocation mechanisms. The pairing phase between apps and devices is not discussed. If having each user define their permissions by hand at initialization is acceptable for a smart home use case, the solution has poor scalability and maintainability.

### 3.4.2 Serverless authorization

Like Fernandez et al. [Fernández et al., 2017], Tamboli et al. [Tamboli and Dambawade, 2016] present a solution where devices are subjects. Like in [Fernández et al., 2017], [Tamboli and Dambawade, 2016] separates authentication and authorization. But unlike [Fernández et al., 2017], [Tamboli and Dambawade, 2016] uses self-descriptive authorization tokens with a built-in time out. Here lies the main difference between the two types of architecture. Once the authorization token has been acquired, no contact with the PDP is needed until it expires. The authentication token similarly carries an expiration date. If the lifetime of the authentication token is longer than the lifetime of the authorization token, the device will not need to re-authenticate. This solution uses Kerberos's tickets [Neuman and Ts'o, 1994] for authentication, CoAP [Shelby et al., 2014] for communication, and ECDSA [Johnson et al., 2001] for signatures. No encryption scheme is proposed, even though token encryption is suggested to protect the subject's privacy. CoAP and ECDSA both have low resource requirements. The authors validate their results with an implementation in a simulated environment.

First presented by Rotondi et al. [Rotondi et al., 2011] and later developed by Gusmeroli et al. [Gusmeroli et al., 2013], the access control solution proposed by IoT@Work, a European project, exploits the delegation support of CapBAC to its fullest extent. Each capability is part of a chain of authorization going back to a root capability. At each stage, a new capability can be created to delegate privileges contained within the parent capability. The delegator specify which rights they are willing to delegate, whether these rights can be further delegated, and how many times. When presenting a capability to the PEP, the subject should include the whole chain so that the PEP can verify the validity of the request. Tokens include a validity period. No description of token issuance is given however. Tokens can be

issued in a distributed fashion, when implementing a Discretionary policy (DAC).

Local policies can be applied to restrict permissions contained within capabilities. This however requires the PDP be involved in every access attempt. The re-introduction of the PDP opens the door for another feature: explicit revocation. The PDP presents an API that can be accessed by capability issuers. They can submit a revocation capability that invalidates a capability they previously issued and all children capabilities that it spawned. Privacy considerations are loosely addressed. Proposed solutions are to encrypt capabilities to protect against eavesdropping, and to self-issue capabilities, using a pseudonym to mask one's identity.

Hummen et al. [Hummen et al., 2014] are focusing on devices too constrained to use public key cryptography. The solution uses built-in features of DTLS (Datagram Transport Layer Security [Rescorla and Modadugu, 2012]) to fall back on symmetric cryptography. Each device is paired with a gateway that acts as the PDP. The gateway and the device are assumed to be operated by the same actor. First, the gateway contacts a subject and establishes a DTLS connection on behalf of the device. Session information are sent to the device, encrypted with the device's master key. When the subject wishes to interact with the device, it resumes the session established by the gateway. The symmetric session key has already been exchanged and no recourse to public key cryptography is required. Several options are discussed for revocation. A first option is for the device to keep a list of sessions that have been revoked. That list is updated by the gateway. In the absence of implicit revocation, the list of active sessions can grow to outweigh the memory available in a heavily constrained device. Moreover, for each incoming connection the device is required to check whether that particular session is on the revocation list, thus increasing the computational cost of establishing a connection over time. A second solution is for the gateway to reset all open sessions. The gateway sends a new session information decryption key to the device. As such, previous session information can no longer be decrypted, thus resetting all the connections. For the sake of privacy, once session information has been pushed to the device, the gateway deletes it. When resetting all the connections, the gateway must then contact all subjects that held a valid session at reset time to start new sessions and generate new session keys.

Ray et al. [Ray et al., 2017] present a solution for Remote Health Care Monitoring. They argue that RBAC is not enough to guarantee patients' privacy. For a given patient, only her doctors should be given access to her data. In case of an emergency, only paramedics at the scene should be given access. A user's role is therefore not enough to determine which information they should be able to access. This is why ABAC, that enables more contextual policies, was preferred. XACML is generally the default choice when using ABAC. The authors choose instead to use NIST Next Generation Access Control (NGAC) [INCITS, 2013, INCITS, 2016], for an easier policy management. The architecture is comprised of 4 types of actors: the monitoring device, the patient's cellphone, the cloud server, and the doctor's device. The monitoring device transmits data by writing to the log file stored on the patient's smartphone. Data is written into either a normal or a critical log. At fixed intervals or when the data is critical, the logs are sent to the cloud where the doctors can subscribe to the feed. The smart phone is in charge of assessing the criticality of the data. The method that it uses is not discussed. It is not clear where the policies are defined and where they are checked. The nature of the data however seems to indicate that serverless authorization should be available.

Ahmad et al. [Ahmad et al., 2018] propose Access Control as a Service. They build their solution on top of AWS IoT extending SecurePG to offer a single, more accessible PAP to enhance administrator

Criteria	[Fernández et al., 2017]	[Bandara et al., 2016]	[Ashibani et al., 2017]	[Hao et al., 2019]	[Yan et al., 2019]
Model	RBAC	RBAC	-	ABAC	-
Crypto	-	RSA	-	CP-ABE, ECC	IBE
Standards	OAuth 2.0	OAuth 2.0, XACM, HTTP, JWT	-	-	-
Use Case	-	Smart Building	Smart Home	-	Smart Home
AuthN or AuthZ	authN & authZ	authN & authZ	authN & authZ	authN & authZ	authZ
Policy storage	independent	independent	on PDP	-	-
Bootstrap phase	yes	no	yes	yes	yes
Resource Efficiency	fair	fair	high	high	high
Resilience	very poor	very poor	very poor	poor	poor
Serverless	no	no	no	no	no
Scalability	poor	fair	poor	poor	poor
Maintainability	fair	high	high	high	high
Permission Updates	imm	imm	imm	-	-
Usability	high	high	high	high	high
Granularity	<i>RL</i>	<i>RL</i>	<i>DL</i>	<i>RL</i>	<i>FL</i>
Context-awareness	very poor	poor	very high	fair	very poor
Revocation	-	<i>impl</i>	<i>impl</i>	-	-
Delegation	-	-	-	-	-
Auditable	-	possibly	yes	-	-
Privacy	poor	poor	very poor	high	high
Governance	single-head	single-head	single-head	multi-head	single-head
Maturity Level	<i>Th</i>	<i>RE</i>	<i>CE</i>	<i>SE</i>	<i>CE</i>

“-”: Not Mentioned or Not Applicable - *DL*: Device Level - *RL*: Resource Level - *FL*: Function Level - *impl*: implicit - authN: authentication - authZ: authorization - *imm*: immediate - *RE*: real environment - *SE*: simulated environment - *CE*: controlled environment - *Th*: Theoretical

Table 3.1: Summary of solutions adopting a centralized architecture without serverless authorization

usability. Policies can be defined using a generic provider-independent language that supports ABAC. Policies are then translated and sent to the Cloud Service Provider (CSP, here AWS). The PDP and PEP function stay within the purview of the CSP, enabling the use of its automated policy verification tools. Authors define 4 architectures to evaluate their proposal. The first two put PDP and PEP in the cloud. The last two entrust the PDP and PEP function to a dedicated edge node that is not the device itself. Architectures 2 and 4 enable stateless functions for unbounded attribute evaluation. Up to 80 attributes have been evaluated in the simulation. The solution is currently only available for AWS IoT but the same principle can be applied to other CSP, thus allowing users to easily choose and change provider. However simulation results show very poor scalability. The underlying framework is mostly responsible as the number of concurrent access is limited by AWS (500) and packets are consistently lost when approaching this upper bound (starting at 330). Additionally, problems linked to the chosen smart lock use case are mentioned that are not addressed by the solution, namely the revocation, logging, and update evasions induced by the absence of connectivity between the end-device and the server, rendering the device dependent on clients to receive policy updates.

### 3.4.3 Conclusion

Table 3.1 and 3.2 show the properties of the different solutions presented in this section. All solutions are at the mechanism level of abstraction. The majority of solutions that do not enable serverless authorization address both authentication and authorization, indicating policies that are most likely to be ID-based and require prior registration. In use cases such as smart homes or smart buildings where the expected number of client is relatively small and behavior can be categorized, the choice of RBAC is relevant. In use cases where actors interact outside of pre-established and fixed patterns however, ABAC and its flexibility are more appropriate as the decision engine is situated in a non-constrained environment. ABAC

Criteria	[Tamboli and Dambawade, 2016]	[Rotondi et al., 2011, Gusmeroli et al., 2013]	[Hummen et al., 2014]	[Ray et al., 2017]	[Ahmad et al., 2018]
Model	-	-	-	ABAC	ABAC
Crypto	ECDSA	-	AES-128	-	-
Standards	CoAp, Kerberos	SAML, XACML	DTLS	NGAC	BLE, AWS IoT, SecurePG
Use Case	-	-	-	eHealth	Smart Lock
AuthN or AuthZ	authN & authZ	authZ	authZ	authZ	authZ
Policy storage	on PDP	on PDP	on PDP	-	Cloud/Device
Bootstrap phase	yes	no	yes	no	yes
Resource Efficiency	poor	poor	very high	very high	very high
Resilience	poor	fair	poor	poor	poor
Serverless	yes	yes	yes	yes	yes
Scalability	fair	fair	fair	fair	poor
Maintainability	high	fair	high	fair	high
Permission Updates	LoT	LoT	imm	-	next access request
Usability	high	fair	poor	high	high
Granularity	AL	RL	DL	RL	RL
Context-awareness	very poor	poor	very poor	high	very high
Revocation	impl	impl & expl	expl	mentioned	impl & expl
Delegation	-	yes	-	-	-
Auditable	-	-	-	no	yes
Privacy	fair	very high	fair	poor	poor
Governance	single-head	multi-head	single-head	single-head	single-head
Maturity Level	SE	implementation	CE	Th	SE

“-”: Not Mentioned or Non Applicable - DL: Device Level - AL: Application Level - expl: explicit - impl: implicit - authN: authentication - authZ: authorization - imm: immediate  
- LoT: Lifetime of Token - RE: real environment - SE: simulated environment - CE: controlled environment - Th: Theoretical

Table 3.2: Summary of solutions adopting a centralized architecture with serverless authorization

might be too demanding for edge intelligence. When serverless authorization is disabled, revoking a permission is equivalent to an update, which is immediately taken into account. With serverless authorization however, means of revoking allotted permissions are necessary. Solutions using self-descriptive capabilities only offer implicit revocation. Indeed in [Gusmeroli et al., 2013], the use of the revocation service requires requests to be examined at access time, thus disabling serverless authorization. Delegation is barely addressed. If the access control decision is taken outside the device, information such as network or request context, i.e. communication protocol, request’s time or subject’s IP, can still be used to qualify the authorization. Note that despite being the architecture best suited for storing access logs, most of the solutions do not address auditability. In case of an incident, it is nonetheless a necessary tool for analysis. A fair proportion of the examined solutions are backed by an implementation. They are however, with the exception of [Bandara et al., 2016], simulating subjects and traffic, sometimes even devices.

### 3.5 Hierarchical Architecture

We define a hierarchical architecture as a set of partially ordered security domains administered by a single entity. That entity may defer the administration of sub-domains to trusted local administrators. Let  $S_i, S_j$  be security domains. If  $S_i \leq S_j$  then policies defined in  $S_j$  supersede policies in  $S_i$ . Security domains can represent a physical or logical separation. Let us examine two examples, both involving smart buildings.

First, to illustrate physical separation, we take the case of a university. The first-level security domain is the whole campus. An example of second-level security domain is the Computer Science department buildings. A classroom or an office would be third-level security domains. Here, the domains represent physical locations. Local administrators all answer to the same head administrator. Policies can be



defined for each level but campus-wide policies supersede department-specific rules: if a student is no longer allowed on campus, they should not be able to open a door or command window blinds in a classroom.

The second example illustrates logical separation with the case of company's offices fitted with many IoT devices with different sensitivity levels. The first-level security domain regroups *public* devices that can be accessed by guests: doors to meeting rooms, light switches, blinds, etc. Doors to restricted area and offices, printers, local temperature controls and others devices reserved for employees belong to a second security domain. Finally, the general temperature control, the server room lock or other sensitive devices belong to a third security domain. Here, contrary to the previous example, a room can host devices belonging to different security domains: a light switch and the temperature control. A user with rights on second-level devices will have rights on first-level devices as well. Several second level domains can co-exist as different teams have access to different devices.

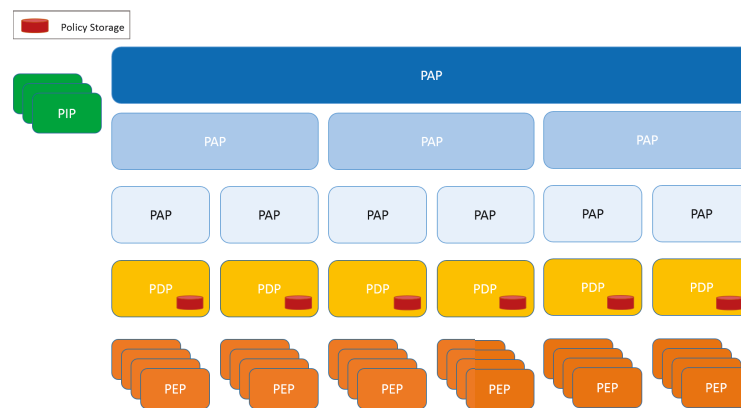


Figure 3.3: Ordered PAP

The partial order of security domains can be translated on either PAP or PDP. In the first option, policies are defined at higher levels and pushed downwards to local PDP that aggregate them. The request is then handled locally according to them. Figure 3.3 breaks down this architecture into functional entities. The head administrator is represented by the top PAP. It pushes policies to all PDP. Below, the other administrators have fewer PDP under their supervision. One PDP still handles several PEP. The PIP can be multiple and common to all PDP or PDP-dependent.

The second option is to have all requests intercepted by a high level PEP and be treated by a chain of ordered PDP going down to the most local one. Policies are not pushed downwards but instead assessed by the PDP of the domain where they have been defined. If a request fails to comply with a PDP's local policies, propagation of the request will stop and a negative response will be sent. Figure 3.4 illustrates the different ways the request can be handled: the request can transit from PDP to PDP (1) or go through PEP at each level (2).

Hierarchical architectures are highly relevant to the IoT: In classical IoT architectures, information are collected by sensors. This information is then gathered by a local gateway that often has more resources than the end-devices. The gateways then send this information to a cloud server and so on.

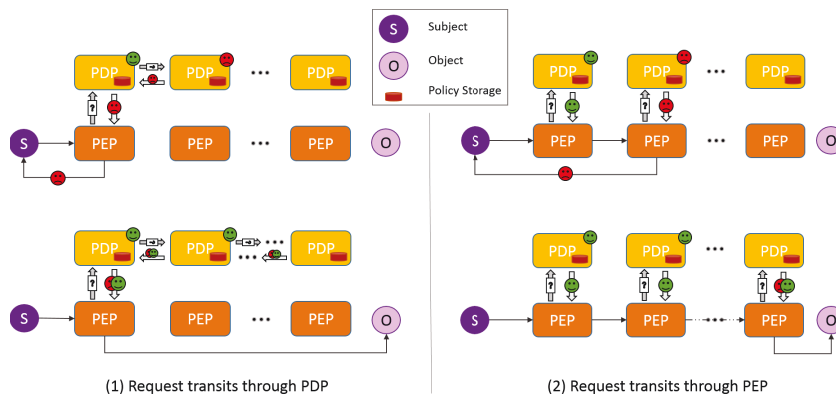


Figure 3.4: Ordered PDP

Such an organization addresses the scalability issue of a centralized system by delegating parts of the administration to more context-aware entities while still leaving the administrative tasks in the hand of a few trusted entities.

It however introduces complexity. Rules must be introduced to handle conflicts between policies that have been designed at different level. Which should take precedence might depend on the request.

**Literature review** Barka et al. [Barka et al., 2015] illustrate the physical separation of security domains. They propose to adapt RBAC [Ferraiolo and Kuhn, 1992] to the Web of Things (WoT), IoT restricted to Web protocols. They divide it into a hierarchy of Ambient Spaces [Mathew et al., 2011], each representing the boundaries of a physical space such as rooms, buildings, neighborhoods, cities, etc. They provide a template to easily organize and integrate new devices into the WoT by means of similarity functions [Mathew, 2013]. Ambient Space Managers (ASM) are entry points into their respective spaces and handle devices and possible embedded Ambient Spaces contained within. They encompass both the function of PEP and PDP, implemented in different modules. Here, devices do not possess access control logic of their own. They are instead accessed through the PEP contained within the ASM of the Ambient Space they belong to. Authors recognize that RBAC is subject to role proliferation and mitigate this risk with role parametrization [Abdallah and Khayat, 2004]. Because the definition of roles is often a static process, we argue that this is not enough to make RBAC usable for the IoT. Revocation mechanisms are not defined.

Ning et al. [Ning et al., 2015] present an authentication scheme for the Unit and Ubiquitous IoT (U2IoT). They consider an industry-oriented scenario with the following actors: IoT Sensors (subjects), IoT Targets (resources), unit Data Centers, industrial Data Centers and national Data Centers. The scheme is made of two protocols: authentication for the Unit IoT and authentication for the Ubiquitous IoT. The first protocol allows the sensor and target to mutually authenticate and be recognized by the Data Center as legal entities. This requires only one message be sent from the target device, and three from the sensor. The second protocol yields mutual authentication of the unit Data Center and the industrial Data Center, both being authenticated to the national Data Center. The scheme ensures

data confidentiality and integrity, forward unlinkability and the privacy of the targets' identities. BAN logic [Burrows et al., 1990] is used to provide a formal analysis of the protocols.

In 1983, Axl and Taylor [Akl and Taylor, 1983] devised a cryptographic solution to key distribution when subjects are organized in partially ordered classes, and sets of permissions are assigned by class. Castiglione et al. [Castiglione et al., 2016] enhance this solution by introducing dynamicity into the system: new subject, reorganization of the hierarchy, revocation of a subject's privileges, etc. The contribution of this paper is two-fold. It introduces both a construction for hierarchical key assignment schemes supporting dynamic updates, using symmetric cryptography as a building block, and a mean to formally prove its security with respect to key indistinguishability. The scheme is well-suited for the IoT as it is made to minimize the impact on subjects. Memory-wise first: each subject only needs to store a single secret value. Computation-wise then: the only operation needed for a subject to recover its key is a single decryption. Bandwidth-wise finally: the only update operation that requires key redistribution is the revocation of a subject. In this instance, a new key needs to be distributed to members of its former class in order for the revoked subject to be unable to decipher messages moving forward. All other updates only impact public information.

Hsiao et al. [Hsiao et al., 2019] similarly build on Axl and Taylor [Akl and Taylor, 1983] by embedding a validity period inside the key, thus enabling implicit revocation. They also use Elliptic Curve Cryptography for its minimal memory requirements.

**Conclusion** Table 3.3 proposes a property summary of the different solutions presented in this section. We first note that the research effort is less significant here than with centralized solutions and yields no implementation. Yet, gateways are often indispensable in IoT scenarios to translate communication protocols and mitigate devices constraints. This naturally translates to a 2-level hierarchy. The potential for hierarchical solutions seems to be undervalued. The solutions presented here cover the whole access control process: first, the actors' bootstrap with key distribution, second, authentication and finally authorization. Delegation of privileges is not addressed, neither is the auditability of the system.

### 3.6 Federated Architecture

A federated architecture is defined by multiple un-ordered security domains operated by different entities interacting. We take the example of two companies collaborating on a smart grid project. Company *A* is an electrical company and has deployed a network of sensors on its area of coverage. Company *B* is tasked with gathering the information and computing usage profiles. Each company has its own security domain but subjects of Company *B* must be able to access objects in Domain *A* in order to operate. Privileges must therefore be granted across domain lines.

Figure 3.5 shows the building blocks of a federated system. Here, subject and object live in different domains. Each domain must then present a PIP accessible to other domains' PDP so that information concerning a subject needs only be stored in one domain but remains accessible to all. Access might also be granted to all subjects providing a proof they belong to a trusted domain. In which case, information must be exchanged before-hand so that the proof can be verified.

The cost of most operations (i.e., new objects, new subjects, access requests) are spread across a

Criteria	[Barka et al., 2015]	[Ning et al., 2015]	[Castiglione et al., 2016]	[Hsiao et al., 2019]
Model	RBAC	-	-	-
Crypto	-	symmetric	symmetric	ECC
Standards	XML	-	-	-
Use Case	-	U2IoT	-	-
AuthN or AuthZ	authZ	authN	authZ (KA)	authZ (KA)
Policy storage	on PDP	-	-	-
Bootstrap phase	no	no	yes	yes
Resource Efficiency	very high	high	high	high
Resilience	poor	poor	poor	poor
Serverless	no	-	yes	no
Scalability	fair	fair	poor	poor
Maintainability	high	poor	fair	fair
Permission Updates	LoS	-	-	-
Usability	high	fair	high	fair
Granularity	<i>RL</i>	-	-	files
Context-awareness	fair	very poor	very poor	very poor
Revocation	<i>impl</i>	-	<i>expl</i>	<i>impl</i>
Delegation	-	-	-	-
Auditable	-	-	-	-
Privacy	fair	very high	high	high
Governance	single-head	multi-head	single-head	single-head
Maturity Level	<i>Th</i>	<i>FA</i>	<i>Th</i>	<i>Th</i>

-: Non Applicable or Not Mentioned - *RL*: Resource Level - *expl*: explicit - *impl*: implicit - authN: authentication - authZ: authorization - KA: Key Assignment - *LoS*: Lifetime of Session - *Th*: Theoretical - *FA*: Formal Analysis

Table 3.3: Summary of solutions adopting a hierarchical architecture

number of security domains, thus providing better scalability. Each domain is free to locally adopt the system that is best suited for it. Administrators control the flow of information and may implement anonymity protocols, providing their users with a different pseudonym for each access. From a user's point of view, a single account can be used to interact with multiple providers. This is great for usability.

Complexity is however introduced by the requirements of inter domain communication that constrain the changes one would want to unilaterally implement. This also introduces the additional issue of controlling access to potentially sensitive subject information. Finally, context emanating from outside the boundaries of the PDP's domain might be harder to get.

**Literature review** Anggorojati et al. [Anggorojati et al., 2012] achieve federation through delegation. In their proposal, CCAAC (Capability-based Context Aware Access Control model), a subject (the delegatee) gains access to a resource outside of its own security domain by using the delegated privileges of a subject from the resource's original domain (the delegator). Each domain must provide an IoT Federa-

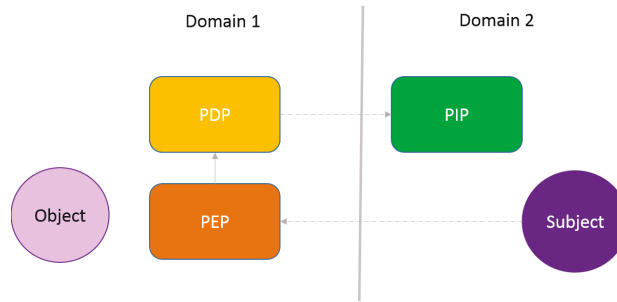


Figure 3.5: Federated architecture

tion Manager to handle delegation requests and set up rules governing inter-domain access. If the request is successful, a pair of internal and external capabilities is generated. The external capability is sent to the delegator that is in charge of transmitting it to the delegatee. It contains the ID of the resource, a set of privileges, and the ID of the delegatee.

An internal capability is also generated and sent to the device hosting the resource. It contains information that will be used to validate the external capability when presented with an access request. The IoT Federation Manager finally updates the propagation tree that keeps track of all delegated privileges. This mitigates one of the main disadvantages of CapBAC, not being able to keep track of the different privileges that have been granted. The external capability is a self-descriptive token that notably contains a *context* field describing conditions to be checked at access time. There are several context types such as time, location or authentication. Each can be evaluated against a constant. If the evaluation fails, access is denied. This improves context-awareness, especially dynamic context that can be checked by the PEP. The descriptivity of the token enables an access process that does not involve the PDP (here the IoT Federation Manager). It however does not include a lifetime and no revocation protocols are described. The delegation process assumes a pre-existing trust relationship between domains and subjects as no discovery mechanism is described to match delegators and delegatees. The system by which a subject requests access to a resource within its own domain is not discussed. No implementation is provided to evaluate the feasibility of the solution.

Xu et al. [Xu et al., 2018b] similarly use capabilities and delegation for federation. It is however the PAP and PDP functions that are delegated from a cloud-based central entity to more local ones, called coordinators, taking advantage of both hierarchical and federated architecture. Periodical synchronization between cloud-based and local PDP ensure policies and profiles stay consistent. The delegation of privilege can be revoked, causing all capabilities issued by a coordinator to become invalid. The new coordinator is tasked with propagating the revocation to all the edge devices. Subject's permissions can similarly be revoked. Capabilities are self-descriptive tokens and carry a *condition* field for access time verification, validity period for implicit revocation, and the public key of the issuer. They are materialized by JSON strings. With multiple PAP, a policy conflict may occur. This possibility is not mentioned by the authors.

Another approach to federation consists in mutualizing resources and entrusting each domain's inter-

domain access policies to a shared entity charged with governing access according to each domain's policies. This is the choice made by Liu et al. [Liu et al., 2017]. They propose a solution to the Authorization Route Optimization Problem (AROP) in RBAC. There are a few problems that an administrator might want to examine to ensure the efficiency of her policies: Safety analysis verifies that no privilege is unintentionally given to an untrusted subject. The reachability problem validates that a specific role can be given to a subject through the actions of the administrators starting from the initial policy. The Authorization Route Planning Problem (ARPP) determines a sequence of actions that the administrators can take to grant subject  $s$  a given set of privileges. The AROP extends on ARPP by trying to provide the best authorization route. As with all optimization issues, the definition of best depends on the metric that is used. Finding a solution to this problem is both about gaining efficiency and therefore scalability, and focusing the roles that should be given to a subject, thus abiding by the principle of least privilege [Schneider, 2003]. This article touches on a subject that often goes unaddressed, namely how administrators grant privileges in the first place.

Sharing of resources is also the option chosen by Uriarte et al. [Uriarte et al., 2016]. This paper presents the Sensing Enabled Access Control (SEAC) concept that merges logical and physical access control. IoT sensors are a gold mine of all sorts of information such as network congestion, a subject's location or a subject's history, and can be integrated to the access control decision process as PIP. The goal is to improve an existing system by increasing fluency and usability while keeping the same level of security. Authorization decisions are taken by the PDP that uses the following sources: the identification system provides physical identification mechanisms such as card readers, the contextual information system gathers and cleans contextual information, the logical access control system provides digital means of authentication such as credentials, the multi-domain security policy configuration system provides domain-related information, the data analysis system's goal is to detect threat and patterns, the geolocation system provides subject tracking across different domains and locations. The solution was evaluated in the port of Valencia. This solution is compatible with legacy system. Its pitfall is the hard balance to strike between usability and security. The inclusion of all these information sources requires a lot of bandwidth and computation to aggregate the data. As this information and its processing are centralized, scalability remains an issue. In a real-time application, security might have to be sacrificed for usability.

Saadeh et al. [Saadeh et al., 2018] propose a solution based on a hierarchical elliptic curve identity based signature protocol to authenticate devices moving around in a smart city and hence jumping between security domains. The scheme involves a single Root Private Key Generator (PKG), several sub PKG, and gateways for each domain. During the initialization phase, public parameters are distributed in the hierarchy and private keys are generated for all parties. Devices are paired with a gateway. A client by selecting two sub PKG in the same security domain and obtains two partial private keys that combine to create their private key. When a client tries to access a device, it transfers the request to its gateway. If the client and the device belong to the same domain, the gateway already possesses all the information necessary to authenticate the client. If not, the home root PKG of the client is contacted through the gateway's hierarchy (sub PKG to root PKG to home root PKG of the client and back). The scheme is analyzed using BAN logic. The signature and subsequent verifications are handled by gateways and PKG. The client and device only need to place one request each. Gateways and sub PKG can be easily added to the system for scalability. The double PKG registration solves the key escrow problem

in which malicious PKG use the private key they generated for the user without its permission (to falsely authenticate a malicious user for instance). PKGs contain an Archive module. However it does not seem to store connection information.

Sciancalepore et al. [Sciancalepore et al., 2018] present a non-canonical use of Cyphertext Policy - Attribute-Based Encryption (CP-ABE): instead of encrypting data, authors propose to challenge requesting subjects with deciphering a random message encrypted with the policy applicable to the requested resource. The scheme includes a single identity authority, and a pair of attribute authority (PIP) and resource server (PEP) per domain. Subjects must first generate an *ephemeral identity* in congress with the identity authority. They prevent the tracking of a client's activity and implement an implicit revocation mechanism as the identity itself has a built-in expiration time. Subjects then authenticate with the relevant attribute authorities with domain-specific credentials to retrieve attributes for their new identity. The ephemeral identity is used to authenticate to the resource server. If the identity is still valid, the resource server encrypts a random message using the requested resource's policy. Only a subject in possession of the required attributes can decipher the challenge. At this point, the resource server may contact the identity authority to check whether the identity is still valid. Identity revocation replaces attribute revocation for the purpose of this scheme. The temporary identity is the main contribution of this proposal, enabling both enhanced user privacy and implicit revocation. Using CP-ABE on a challenge message furthers user privacy as their attributes do not need to transit through the network. Serverless authorization is available though it cannot be used while explicit revocation is enabled. The requirement for users to register with each attribute authorities individually is a blow to usability. This results in high bandwidth and storage requirements for the subject and scalability issues as clients are not divided amongst domain but duplicated. We also note that although the privacy of users is protected, information can be leaked by studying the cyphertext policies sent to users.

Criteria	[Anggorojati et al., 2012]	[Xu et al., 2018b]	[Liu et al., 2017]	[Uriarte et al., 2016]	[Saadeh et al., 2018]	[Sciancalepore et al., 2018]
Model	-	-	RBAC	RBAC	-	ABAC
Crypto	-	asymmetric	-	-	ECC	CP-ABE
Standards	-	JSON	-	-	-	JWT, TLS, X.509
Use Case	-	-	Industry 4.0	-	Smart City	-
AuthN or AuthZ	authZ	authZ	authZ	authZ	authN	authN & authZ
Policy storage	on PDP	on PDPs	independent	on PDP	-	on PDP
Bootstrap phase	no	no	no	no	yes	yes
Resource Efficiency	poor	poor	very poor	very poor	high	poor
Resilience	high	high	poor	poor	poor	fair
Serverless	yes	yes	-	no	no	yes
Scalability	high	very high	poor	poor	high	poor
Maintainability	high	high	fair	fair	high	fair
Permission Updates	LoT	LoT + sync	-	imm	-	imm/Lold
Usability	fair	high	high	high	high	poor
Granularity	RL	RL	-	DL	-	RL
Context-awareness	high	high	-	very high	-	high
Revocation	-	impl & expl	-	-	-	impl & expl
Delegation	yes	of PAP/PDP	-	-	no	-
Auditable	-	-	-	yes	no	suggested
Privacy	poor	poor	-	poor	fair	high
Governance	multi-head	multi-head	multi-head	multi-head	multi-head	multi-head
Maturity Level	Th	SE	Th	RE	FA	SE

“-”: Not Mentioned or Non Applicable - *DL*: Device Level - *RL*: Resource Level - *expl*: explicit - *impl*: implicit - *authN*: authentication - *authZ*: authorization - *imm*: immediate - *LoT*: Lifetime of Token - *Lold*: Lifetime of Identity - *RE*: real environment - *SE*: simulated environment - *Th*: Theoretical - *FA*: Formal Analysis

Table 3.4: Summary of solutions adopting a federated architecture

**Conclusion** Table 3.4 shows the properties of the different solutions presented in this section. Both Xu et al. [Xu et al., 2018b] and Saadeh et al. [Saadeh et al., 2018] present solution that use hierarchical elements. As for hierarchical architecture, federation is not the focal point of the research community. Collaboration is however bound to be an essential part of the IoT. Devices from different manufacturers acting on behalf of different companies will have to work together to accommodate all use cases.

## 3.7 Distributed Architecture

In opposition to centralized systems where a single PDP serves all requests, in distributed systems the PDP function is performed by several entities. We identify three families of distributed architectures.

In the first family of solutions, the PEP takes on part of the PDP function. This usually translates to a semi-distributed model where a central entity takes part of the access control decision and leaves the rest to the device itself. Those solutions are discussed in Section 3.7.1.

In the second family of solutions, several PDP are accessible to each PEP, potentially simultaneously. Section 3.7.2 presents these proposals.

Finally, the third distributed architecture is based on the blockchain, where policies are stored. The miners will often act as PDP by approving transactions representing access requests. These solutions are detailed in Section 3.7.3.

### 3.7.1 PDP/PEP hybrid

Our first approach to PDP distribution is to leave part of the decision to the PEP. We will consider the PEP to be included within the device itself. Authorizations are granted by the PDP under some conditions to be evaluated by the device at the time of access. The device is no longer just a PEP and can choose to allow or deny access if the conditions communicated by the PDP are not met. Figure 3.6 illustrates this process.

The bigger advantage here is context-awareness. By placing the final decision as close as possible to the object, local context can factor into it. The delay between condition evaluation and access is considerably reduced. Conditions can be pushed from the PDP to the PEP using tokens. In which case, serverless authorization is supported (see Section 3.2.2).

The impact on the device however, is significant. That type of solutions cannot be used with strongly constrained devices. Computing resources are necessary to evaluate the local conditions. Furthermore, this evaluation must be performed with every access, to be as up-to-date as possible, or the benefits of context-awareness are lost. These solutions are not as scalable as multi PDP solutions.

**Literature review** Cerf [Cerf, 2015] provides no solution to the access control problem but rather directions for future research that fit the hybrid architecture. The author proposes to move the AC logic to the edge of the network by using capabilities. Each device is paired with a unique *controller* that will be its only interlocutor. The device produces a capability consisting in a nonce encrypted with the controller's public key and signed with the device's private key. When communicating, the nonce must be returned encrypted with the device's public key and signed with the controller's private key. It is single use. It is interesting to note that here the end-device is both a PDP and a PEP but not simultaneously.



Criteria	[Hemdi and Deters, 2016]	[Bernabe et al., 2016]	[Hussein et al., 2017a]	[Hernández-Ramos et al., 2013]	[Cerf, 2015]	[Seitz et al., 2013], [Cherkaoui et al., 2014]
Model	ABAC	ABAC, TBAC	ABAC	-	-	-
Crypto	-	ECC	-	ECDSA	asymmetric	AES
Standards	CoAP	CoAP, JSON, XACML	CoAP, JSON	CoAP, JSON	-	CoAP, DTLS, XACML, SAML, JWE
Use Case	-	-	-	-	-	-
AuthN or AuthZ	authZ	authZ	authZ	authZ	authZ	authZ
Policy storage	on device	on PDP	on PDP	-	on device	on PDP
Bootstrap	no	no	no	no	no	yes
Resource Efficiency	very poor	very poor	poor	very poor	very poor	very poor
Resilience	fair	high	high	fair	fair	poor
Serverless	no	yes	yes	yes	no	no
Scalability	poor	high	very high	fair	poor	fair
Maintainability	poor	fair	high	high	poor	fair
Permission Updates	imm	LoT	LoT	LoT	NTR	NTR (single use)
Usability	poor	fair	high	fair	poor	high
Granularity	-	RL	RL	RL	DL	RL
Context-awareness	fair	very high	high	high	very poor	high
Revocation	-	impl and expl	impl and expl	impl	-	impl
Delegation	-	-	-	-	-	-
Auditable	yes	-	-	-	-	-
Privacy	-	fair	poor	very poor	high	poor
Governance	single-head	single-head	multi-head	-	single-head	multi-head
Maturity Level	SE	SE	SE	SE	Th	SE

“-”: Non Mentioned or Non Applicable - *DL*: Device Level - *RL*: Resource Level - *expl*: explicit - *impl*: implicit - *authN*: authentication - *authZ*: authorization - *LoT*: Lifetime of Token - *SE*: simulated environment - *NTR*: Next Token Request - *Th*: Theoretical

Table 3.5: Summary of solutions adopting an hybrid distributed architecture

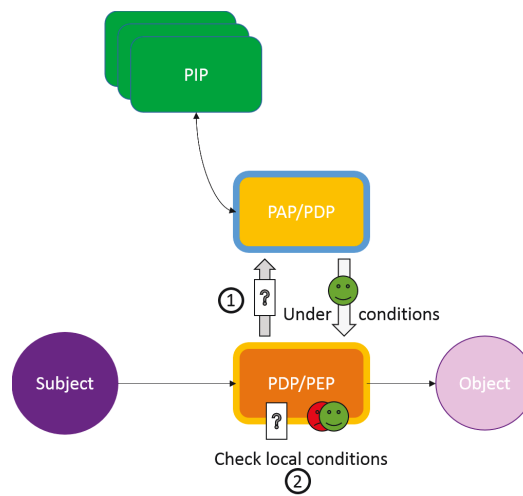


Figure 3.6: Access request in the hybrid architecture

The device must indeed decipher and check the validity of the nonce previously generated by itself. The capability is opaque.

A few hypothesis are made on the devices' abilities: they can generate their own keys, execute cryptographic operations (symmetric or asymmetric) and resist physical tampering. Even when the controller is assumed to change IP over time, it should still be possible to associate it with a unique identifier such as a domain name. The association of the device with a unique controller departs from the classical postulate that in the IoT, the list of subjects wishing to access a given object is dynamic, or at least large.

Seitz et al. [Seitz et al., 2013] present a generic authorization framework using JWE [Jones and Hildebrand, 2015] tokens. XACML [Moses et al., 2005] policies are pushed to the PDP by the device owner. Subjects can use a discovery system to request their token directly from the PDP. The appropriate rule is partially evaluated by the PDP, leaving only local conditions to be verified. Session keys may be delivered with the tokens to encrypt messages. To prevent replay attacks, the tokens can only be used for a short time after issuance and only once. The device may also store recently used tokens. Consequently permission updates take effect with only a small delay. This however represents an increase in the number of required messages for subjects as each access require an exchange with the PDP as well as an access request to the PEP. The solution is replay-resistant and requires both ends to mutually authenticate.

This solution is used as a frame of reference by Cherkaoui et al. [Cherkaoui et al., 2014] that focus on the physical aspect of the IoT. It presents Physical Unclonable Functions (PUF) [Pappu et al., 2002] as a mean to generate and protect the keys used to secure communication. PUF uses the unique physical characteristics of a device to produce a secret key that is unique, random, reliable and tamper resistant. PUF are also a low cost solution. eSIM are introduced as a way to mitigate the discrepancy between the life expectancy of a SIM and its IoT device. This article brings a solution to issues such as key provisioning or resistance to physical tampering that appeared as requirements in [Seitz et al., 2013] but were not solved.

Hernandez-Ramoz et al. [Hernández-Ramos et al., 2013] propose a solution bearing a lot of similarities with [Seitz et al., 2013]. The capability however can be used multiple times and carries an expiration date for implicit revocation along with local conditions. Cryptographic operations are performed last to minimize the computational burden on the device. The request is transmitted using CoAp [Shelby et al., 2014]. Resources are designated by a unique URI. The process of issuing the token is unfortunately considered out of scope.

Hemdi et Deters [Hemdi and Deters, 2016] propose to use attributes to identify users rather than passwords or signatures alone as they consider users to be irresponsible with their credentials. By combining numerous attributes - username, password, MAC address, access time, etc - the likelihood of identity theft is considerably lowered. The emphasis is put on context. Their solution is mainly presented through its implementation. Special care has been taken to lower the computational burden on the device.

The subject is a web application that uses CoAP to push its requests, in a JSON format. The request is sent to a server running on a Raspberry Pi 2 that exposes a RESTful service. Parsed attributes are then compared with data stored in a MySQL database. All requests, successful or otherwise, are logged in the database. If the request is valid, the server responds with a *Hello World* message. The only actors here are the subject and the device. The latter then acts as both the PEP and PDP.

Here the access control logic is left entirely in the hands of the end-device. Enrollment of new users and their devices (MAC address) is not addressed, neither is the provisioning of permissions or their granularity. Revocation of attributed permissions or deletion of users is not discussed. Support for delegation is not mentioned. Because the access control decision is taken on the device itself, there is a potential for a high level of context-awareness, at least on the side of the end-device. However, the example implementation does not embrace this potential. The device does keep track of access attempts. But because of its limited storage capacities, a big number of failed attempts might fill its memory and hinder its ability to function normally, opening a new attack vector. Finally, because the end-device itself needs to be configured to accept new clients, the solution would not scale. Maintenance might require the end-device to be made inaccessible which would interrupt the service. Users would need to not only register with every end-device but also to register every new device used to access the service with every one of them, making for poor usability. The authors have evaluated their solution by firing a number of requests (25, 50 and 100) in a short time span (125 ms, 250 ms and 500 ms), using a varying number of user (5 to 15 users). These numbers are not on scale with the load expected of most IoT systems.

Bernabe et al. [Bernabe et al., 2016] put trust at the center of the access decision. Several devices are able to perform the same task. A discovery system allow subjects to obtain a list of equivalent devices and contact the most trusted. Trust is calculated based on quality of service (availability, delay, etc), security (communication protocol, network, etc), reputation according to other devices, and social relationship [Atzori et al., 2012]. On constrained devices, the trust value is computed using fuzzy logic. The Distributed Capability-Based Access Control framework (DCAPBAC) [Hernández-Ramos et al., 2016] is used to implement CapBAC. Tokens carry an expiration date and local conditions including the trust value the device should associate to the subject. Policies are modeled using ABAC and expressed with XACML.

Following along the lines of Social IoT, Hussein et al. [Hussein et al., 2017a] introduce COCapBAC, for COmmunity-driven Capability-Based Access Control. IoT devices are grouped to form communities

of heterogeneous devices that share a common goal. Within the community, the following roles are distributed according to devices' capacities: PIP certifying attributes, PDP issuing capabilities, and PEP in charge of their validation. The latter are called Gatekeepers. Subjects contact PIP to get their attributes certified and send them to the PEP along with their token request. The PEP forwards the request to the PDP and returns the decision and eventual token containing a expiration time and potential conditions to the subject. When accessing an object, it is not clear whether the subject should direct its request to the Gatekeeper or to the device that holds the object. In the latter case, the device forwards the request to the Gatekeeper. In the case where local conditions should be checked, the Gatekeeper acts as both the PDP and PEP.

The solution has been implemented by the authors. If the format of the token and its issuance have been described in details, the process of forming the community, role attribution, attributes verification, delegation mechanisms or policy definitions are not discussed. In this solution, the subject does not interact with the PDP. The number of messages exchanged by the PEP is therefore increased. The subjects can be assumed to be devices themselves. Revocation of granted privileges can be achieved implicitly by setting an expiration time in the token or explicitly by pushing revocation policies to the Gatekeeper.

### 3.7.2 Multi PDP

A simpler approach to PDP distribution is to multiply the number of PDP. This can be achieved by either letting the PEP be registered with several PDP at once or by providing an easy registration process that lets the PEP switch PDP effortlessly, providing the PDP has rapid synchronization.

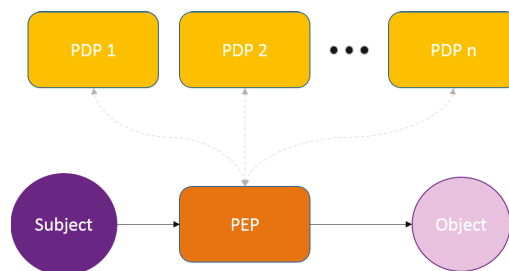


Figure 3.7: Multi PDP architecture

The first benefit is increased resilience. Indeed, if a PEP's resources were registered with several PDP simultaneously, requests can be redirected in case one of them is compromised or unavailable. Similarly, resources can be registered with a new PDP if the configuration cost is minimal, though a delay might be introduced. In the case where resources on the same device are split amongst different PDP, the PEP will continue to be at least partially operational, making it more resilient. Splitting objects amongst several PDP can hide the PEP's purpose, thus increasing privacy.

Scalability is also increased as the load is balanced amongst several PDP instead of one. This means each PDP can be taken offline with a limited incidence on operations. Maintenance is therefore made easier.

Multiplying PDP however introduces complexity in the following ways. First access control policies have to be replicated across the different PDP. When a permission needs to be updated, propagation may cause a significant delay. Asynchronous updates are required for PDP that were offline. Second, a discovery system should be implemented. This system can be used either by the subjects that wish to know which PDP they should contact, or by PEP to find new PDP available for registration. In addition, this method does not increase context-awareness as decisions are still taken outside of the device.

Criteria	[Wu et al., 2017]	[Fotiou et al., 2016]	[Patel et al., 2016]
Model	-	-	-
Crypto	-	symmetric	ECC
Standards	-	CoAP	-
Use Case	Smart Home	-	-
AuthN or AuthZ	authZ	authZ	authN & authZ
Policy storage	on PDP	on PDP	on PDP
Bootstrap phase	yes	yes	no
Resource Efficiency	high	fair	very high
Resilience	fair	very high	very high
Serverless	yes	yes	yes
Scalability	fair	very high	high
Maintainability	poor	high	high
Permission Updates	LoT	LoS	LoT
Usability	high	poor	fair
Granularity	RL	RL	RL
Context-awareness	poor	poor	-
Revocation	impl	impl	impl
Delegation	-	-	yes
Auditable	-	-	-
Privacy	fair	high	poor
Governance	multi-head	multi-head	multi-head
Maturity Level	SE	SE	Th

“-”: Non Mentioned or Non Applicable - *RL*: Resource Level - *expl*: explicit - *impl*: implicit - authN: authentication - authZ: authorization - *LoT*: Lifetime of Token - *SE*: simulated environment - *LoS*: lifetime of a session - Int BC: integration in the blockchain - *Th*: Theoretical

Table 3.6: Summary of solutions adopting a distributed architecture with multiple PDP

**Literature review** Wu et al. [Wu et al., 2017] splits the role of the PDP between two entities: the gateway and the device’s owner. When trying to access a resource, the subject first contacts the gateway that examines the request and returns the address of the owner and a counter number. This pre-approval step protects the owner’s privacy and works as a discovery system. If the owner grants the request, they return a key  $k_1$  to the subject, and sign the provided counter number. The subject exchanges the signed counter number against a key  $k_2$ . Concatenating  $k_1$  and  $k_2$ , the subject obtains a key the session key  $k$  used to communicate with the device. A counter is used in lieu of a lifetime or expiration date

as IoT devices cannot be assumed to have access to a synchronized clock. In these first steps, the number of exchanged messages is important but none involve the device. The subject then forms its own authorization token out of information retrieved from both the gateway and the owner, encrypts it with  $k$ , and attaches it to its access request. The device deciphers the token, and validates the information within.

Fotiou et al. [Fotiou et al., 2016] provide re-usable generic policies. Examples include allowing access to all subjects belonging to a given company or making their request from a given location. Policies are registered with a specific PDP. When configuring their devices, owners choose the policies that are best suited for their resources. They then collect the URI of the desired policies and contact the corresponding PDP to obtain a secret key generated from the device's identity. Keys and URI are provisioned to the device. To gain access to a resource, a subject sends an unauthorized request to the device containing their ID - one per PDP. The device responds with an opaque token and the policy URI, and derives its own symmetric key from the subject's ID only using a HMAC. The subject determine which PDP it should contact using the URI. The request it sends is PDP-specific but should contain the subject's ID, the device's ID, the URI policy and the token. The PDP authenticates the subject and evaluates its request. If access is to be granted, the PDP derives a key that should match the one derived by the device and can be used to open a secure communication channel. The device will still check that the resource in the authorized request matches the one used to obtain the session key and if the validity of the session key has not expired. Once a transaction is completed, the key is removed from the device's memory. Every token is single-use. Different identifier can be used for each unauthorized request to protect user privacy in regards to the device and any eavesdropper. The generic nature of policies enables information about the subject to be hidden from the PDP. Alternatively, a resource can be associated to an ordered list of policy URI. When making the authorized request, the subject should specify which URI was used. The key can be derived from there.

Patel et al. [Patel et al., 2016] propose a mutual authentication protocol using a third party, called the Coordination Node (CN). It is used by a subject  $s$  wishing to connect to a device  $d$ . The protocol uses elliptic curves as they allow higher level of security with smaller keys. It is designed to minimize the burden on  $d$  that is the most constrained of the three actors. The CN acts as a discovery system and can hide changes in network topology. Prior to engaging in mutual authentication, the subject should acquire a capability from a PDP. No details are given on the rules governing the issuance of that capability. The token is self-descriptive and, besides an issuance and expiration time, it contains a delegability bit, indicating whether the permissions within can be delegated to another subject, even though no delegation mechanism is presented. A secret key is also generated and sent to  $s$  via a secure communication channel. The authors use formal verification tools, namely AVISPA [Armando et al., 2005], to validate their mutual authentication protocol.

### 3.7.3 Blockchain-based solutions

The blockchain is a recent addition to the portfolio of decentralization tools. The underlying blockchain technology is the subject of Chapter 2. A number of solution explore its potential in access control situations. Figure 3.8 illustrates the most recurring architecture. PAP push policies to the blockchain where they are stored. Miners act as PDP by validating transactions. Devices act as PEP. They connect

to the blockchain via a gateway.

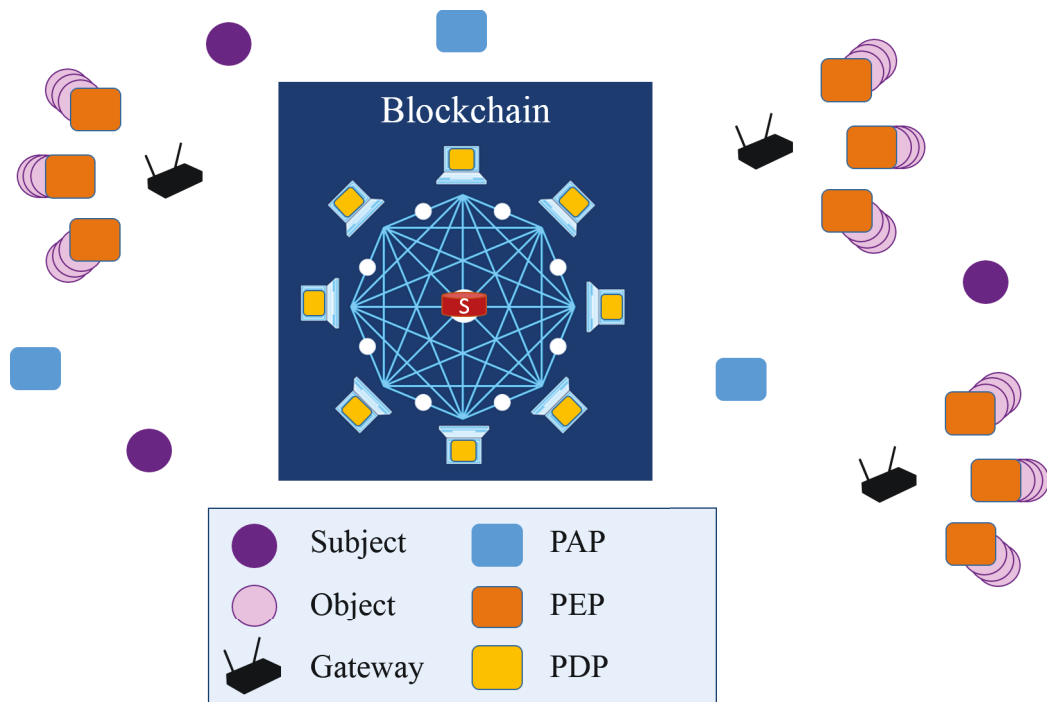


Figure 3.8: Blockchain-based architecture

The blockchain can be used to store access rules, tokens, past transgressions, etc, thus providing auditability. The availability of smart contract source code provides transparency regarding the access decision process and the handling of information. This can cause a privacy issue however depending on the blockchain model. Permissioned blockchain should be preferred when dealing with sensitive information. By nature, the blockchain is a decentralized trustless network. This translates to an increased resilience as any node can be contacted to retrieve the content of the ledger. Blockchain credentials can be generated by users independently. This circumvents the key escrow problem: when a private key is generated by a third party, users have to trust that said party will not pose as themselves or sell their information. This problem is all the more important when a single entity holds the keys of a majority of the participants. Actors can use the same keys and identities regardless of the PDP they address. This greatly simplifies the configuration process. The blockchain protocol handles propagation of permission updates.

On the flip side, the resource-demanding nature of the blockchain architecture leads to a number of problems: how can we connect constrained devices to the blockchain without re-introducing centralization? What level of trust can be put into the information extracted from the blockchain when the device cannot possibly run a full node? With variable transaction validation time, how apt is the blockchain to tackle time-sensitive use cases? The blockchain is a new paradigm that may yet lack the maturity

required to tackle these issues. Also in question, the usability and user-friendliness of existing solutions. The scalability of the blockchain itself remains a point of contention. As a result, the type of information hosted and the specifics of the underlying blockchain have a strong impact on scalability. The number of required smart contracts for instance can drastically drive up the cost of a solution. Maintenance is also a concern: once deployed, smart contracts cannot be modified unless they were designed to be, new contracts are given new blockchain addresses that need to be communicated to devices that are already deployed, changes to the blockchain require community adoption and may not be backward compatible, etc.

These issues are explored in details in Section 2.9 and Section 2.10.

Criteria	[Zyskind et al., 2015]	[Ouaddah et al., 2016a], [Ouaddah et al., 2017a]	[Novo, 2018]	[Pinno et al., 2017]	[Xu et al., 2018a]	[Zhang et al., 2018b]
Model	-	-	-	-	-	-
Crypto	encrypt: symmetric, sign:ECDSA	ECC	asymmetric	asymmetric	-	-
Standards	Kademilia, LevelDB	Bitcoin	CoAp, JSON-RPC	-	Ethereum, JSON, SQLite	Ethereum
Use Case	Mobile user	-	-	-	-	-
AuthN or AuthZ	authN & authZ	authN & authZ	authZ	authZ	authN & authZ	authZ
Policy storage	on BC	on BC	on BC	on BC	on coordinator, on Device	on BC
Bootstrap phase	yes	yes	yes	no	yes	yes
Resource Efficiency	poor	poor	high	high	fair	fair
Resilience	very high	very high	very high	very high	very high	very high
Serverless	no	yes	no	-	no	yes
Scalability	very poor	high	poor	very high	fair	fair
Maintainability	high	high	high	high	fair	high
Permission Updates	Int BC	Int BC	Int BC	Int BC	Int BC	Int BC
Usability	high	fair	high	fair	very high	high
Granularity	RL	RL	RL	RL	RL	RL
Context-awareness	very poor	high	very poor	very high	fair	fair
Revocation	expl	expl	impl & expl	-	impl & expl	-
Delegation	no	yes	-	-	yes	-
Auditable	yes	yes	yes	yes	yes	yes
Privacy	very high	fair	poor	poor	poor	poor
Governance	multi-head	multi-head	multi-head	multi-head	multi-head	multi-head
Maturity Level	Th	SE	SE	Th	SE	SE

“-”: Non Mentioned or Non Applicable - *RL*: Resource Level - *expl*: explicit - *impl*: implicit - *authN*: authentication - *authZ*: authorization - *LoT*: Lifetime of Token - *SE*: simulated environment - *LoS*: lifetime of a session - *Int BC*: integration in the blockchain - *Th*: Theoretical

Table 3.7: Summary of solutions adopting a blockchain-based architecture

**Literature review** Zyskind et al. [Zyskind et al., 2015] were the first to use the blockchain for access control purposes. Their paper, though not specific to the IoT, aims at ensuring the privacy of a mobile user’s data. Encrypted data are stored off-blockchain whereas the access rules are published to the blockchain. Users have unlimited access to their own data and can modify or revoke granted permissions by publishing a blockchain transaction. Applications wishing to access the data publish a request to the blockchain, leaving a visible trace of every access request they make. This request is compared to the stored policy, consisting in a list of resources a given service is able to access. Users can generate as many pseudonymous identities as they wish in an independent fashion to protect their privacy. A compound identity must be created for each user/service pair with an associated encryption key. This means that if ten services require access to the same data, the user not only has to generate ten compound identities but encrypt the same data ten times and send ten blockchain transactions, storing ten copies. Though this is great for privacy, it is terrible for scalability.



Ouaddad et al. [Ouaddah et al., 2017a] present a more feasible solution. Subjects must contact the device owner that then publishes a blockchain transaction detailing the conditions under which access is to be granted to the subject. Another blockchain transaction must be issued by the subject to prove its compliance with the prescribed conditions and obtain a token. These operations must be repeated before each access, leading to a slow process, especially considering the validation time for blockchain transactions that range from an average of 10mn in Bitcoin to 12s in Ethereum. Validation time also highly fluctuates over time and can depend on the fee associated to the transaction. The authors present an implementation of their solution in [Ouaddah et al., 2016a].

Novo [Novo, 2018] uses a single smart contract to store the policies of all devices in his system. This may cause scalability issues. IoT devices are paired with one or several PAP (managers) and access the blockchain through *management hubs*. Retrieving policies at access time is a simple *read* operation that does not suffer the limitations of blockchain transactions but still requires a live blockchain access. The article focuses on feasibility rather than security so some issues such as revocation or bootstrap have suggested solutions and are not included in the implementation.

Pinno et al. [Pinno et al., 2017] present a system that can accommodate ACL, attribute-based and token-based schemes. Four types of information are registered on the blockchain: context, relationships, rules, and accountability information. The context can either be registered automatically by sensors or manually inputted. Relationships are a link between two entities with optional attributes. An example would be a link between User A and User B with a *friend* attribute. Entities can be users, devices, or groups. Each must be linked to an owner. When an access request is placed, the appropriate rules are consulted. Besides access conditions, rules also define the information that is to be written into the Accountability blockchain, whether the access was a success or a failure. Policies can therefore be written to take into account the previous behavior of subjects, relationships between entities, and contextual information.

Xu et al. [Xu et al., 2018a] implement a capability-based solution using smart contracts, with a focus on delegation as a means of scaling administration. Authors present a federated architecture with a PDP (coordinator) for each domain, as well as a centralized cloud server to handle cross-domain delegation. Conditions for delegation can be specified, to guarantee separation of duty for instance. Users register with their domain coordinator that also handles access requests. When a request is granted, the associated token is sent to a smart contract. Subjects produce the smart contract address when trying to access a device. Acting as the PEP, the device contacts the blockchain to retrieve the token. It may contain additional conditions that will be verified by the device as well as an expiration time. When a capability is revoked, all delegated privileges that spawned from this capability are revoked as well. Any subject up the chain can revoke a children capability. No description of initial capability issuance is provided. In particular, the authorization model is not discussed. So while this solution offers interoperability and high scalability, the potential depth of the delegation tree makes it hard for administrators to follow and manage authorizations, though depth and width of authorization can be defined and adjusted by delegators. Authors forbid two capabilities to be delegated to a single subject simultaneously to maintain a tree structure instead of a Directed Acyclic Graph (DAG). The complexity of administration is only slightly reduced while this is an important constraint on operation.

Zhang et al. [Zhang et al., 2018b] use smart contracts to encode Access Control Lists onto the blockchain. One contract is required per resource/subject pair and per authentication method. The poli-

cies are therefore static and hyper specific. They can contain negative permissions, i.e. an ACL can *deny* a request for a given subject, object, action triplet. Subject behaviors are recorded and can factor into the authorization decision. Given examples of misbehavior are subsequent access failure, or too many requests in a given time frame. Special smart contracts called *Judge Contracts* are deployed and linked with ACL contracts to punish bad behavior, by banning offending subjects for a limited time for instance. The addition of a new user or object requires the deployment of a number of smart contracts, which takes time, computation, and space. Updating permissions require blockchain transactions as well. Similarly to [Novo, 2018] however, retrieving an ACL is a simple read operation. A single smart contract maintains a lookup table containing all the information to find the appropriate smart contract for each access request: address, creator, associated subject and object. This simplifies maintenance operation as actors need only be configured with one smart contract address. The size of this contract however might hinder scalability.

### 3.7.4 Conclusion

Table 3.5 presents the properties of solutions using a PDP/PEP hybrid. The solutions presented here are generic and do not apply to a specific use case. They resemble centralized architectures with added context-awareness and represent the lion's share of the research efforts regarding access control solutions for the IoT. As such, they present many similarities. ABAC is preferred here for its ability to factor contextual information into the access decision. Resources are mostly REST API and actions correspond to CoAP methods. When tokens are used, which is the case for half of the solutions and equate to serverless authorization support, they are implemented using JSON. Almost all solutions yield an implementation. It is regretful then that they do not provide more details about the bootstrapping process. Interestingly, solutions using serverless authorization still offer explicit revocation. Delegation and auditability however are not addressed.

Table 3.6 shows the property summary of the different solutions using multiple PDP.

The majority supports serverless authorization. This seems natural as the rationale behind these solutions is to be only loosely attached to any single PDP. The downside however is that the revocation of allotted rights is implicit. Policies are fine-grained. Most of them provide details on the bootstrapping process. Contrary to other architectures, the bootstrap phase is likely to be repeated multiple times over the course of the device life. These details are therefore important to be able to evaluate the true impact of the solution. Implementations demonstrate the feasibility of such solutions. Delegation mechanisms are barely addressed.

Table 3.7 shows the property summary of the different blockchain-based solutions.

In all of the presented solutions, access policies are stored on the blockchain. This increases resilience and maintainability but compromises privacy. Indeed, most blockchain implementations are only pseudonymous. Studies [Reid and Harrigan, 2013] have shown that passive observation of the network coupled with external information could lift the pseudonymity of users. Even with permissioned blockchains, special care must be put into deciding what information gets recorded. The burden of privacy protection falls on subjects that can cover their tracks by using multiple identities. Gateways are used to connect IoT devices to the blockchain. When they are blockchain nodes themselves, the system enjoys semi-serverless authorization that we call *local access*: only a connection to the local gateway is

required at access time. This connection is likely to occur over short range protocols, likely the same that are used by subjects. Hence if the connection to the gateway is disrupted, the connection between client and device is also likely to falter. The unavailability of the access control system is then not the biggest issue. Updates are all dependent on transaction validation. This time varies greatly from blockchain to blockchain and sometimes from moment to moment. Only a few solutions offer implicit revocation. Blockchain-based solutions are therefore not indicated for systems that require timely revocation. Resource efficiency and context awareness are highly dependent on the specifics of each solution.

## 3.8 Taxonomy & Analysis

### 3.8.1 Analysis

From the work presented in Sections 3.4 to 3.7 the following conclusions can be drawn.

The bulge of the research effort is focused on centralized solutions. Second come distributed proposals. Note that most solutions that fall under our PDP/PEP hybrid architecture use a centralized PDP and therefore also suffer from the single point of failure and scalability issue. Recently, the emergence of the blockchain has led to a surge in blockchain-based AC solutions that aim to remedy this. There also seem to be an increased interest for federated architectures. However, hierarchical solutions that could alleviate the load on a single central PDP and therefore increase the scalability of centralized solutions are not being explored.

Furthermore, looking at the degree of diversity between the solutions within each architecture, the centralized and PDP/PEP hybrid proposals present the most similarities. Having been studied the most, they are crystallizing when multi PDP, federated, and hierarchical solutions present more opportunities for creativity and improvements. Federation for instance can be combined with hierarchical signature scheme [Saadeh et al., 2018] or PDP/PEP local condition evaluation [Xu et al., 2018b].

Revocation mechanisms are missing from many solutions. Serverless authorization seems to be incompatible with explicit revocation as the connection to the server needs to be active to check for revoked privileges and attributes [Gusmeroli et al., 2013, Ahmad et al., 2018, Sciancalepore et al., 2018]. In the IoT context where devices are physically vulnerable, operate without supervision, and are difficult to update, no access control solution should be deployed without support for revocation.

Two other features are scarce in the literature: auditability and support for delegation. Auditability is essential for forensic purposes. Numerous IoT use cases require devices to operate without supervision. Logs are then the only tools at our disposal to study potential incidents. Delegation, though not as essential, seems adapted to use cases such as smart homes that operate under DAC. With proper configuration, it constitutes a good way to increase usability and decrease administrators' load.

### 3.8.2 Taxonomy

Table 3.8 presents a summary of each architecture qualitative strength and weaknesses. Some of the criteria defined in Section 3.3, such as the granularity of permissions or maturity level, are only relevant to the assessment of full-fledge proposals. They are therefore not included in the table.

Criteria	Centralized		Hierarchical	Federated	Distributed		
	Serverless	Not serverless			multi PDP	Hybrid	Blockchain
Resource Efficiency	+	++	+	+	-	--	--
Resilience	-	--	+	+	++	-	++
Scalability	-	--	+	+	++	-	+
Maintainability	++	+	-	--	--		+
Permission Updates	-	++	+	-	--	+	-
Usability	++	+		++	-		+
Context-awareness	--	-	+			++	
Revocation	--	++	++	-	--		+
Auditable	+	++		-	--	-	++
Privacy	--	--		++	++	+	-
Governance	++	++	+	--	--	+	--

Table 3.8: Comparison of architectures

We use the most differentiating criteria from Table 3.8 to build our taxonomy, illustrated in Figure 3.9. Scalability and Resilience for instance are both good candidates for a first-level separation as they split our architectures roughly in two. When two criteria can be used, we take the most relevant for the IoT context or with regard to the architectures.

### 3.8.3 Future research direction

Based on the state of the art presented in this chapter, we believe the following challenges should be tackled by future proposals:

**Hybrid architectures** Most access control solutions deployed today rely on centralized PDP. This leads to scalability and context-awareness issues. Edge intelligence is limited by resource constrains. However these two extremes are not the only options. Hybrid architectures - hierarchical, federated, multi PDP - enable a compromise between requirements and limitations.

**Usability** To be efficient, security measures must be used. If they hinder the user experience, access control solutions are likely to be de-activated or circumvented by end users. Designing solutions with usability in mind will lead to a faster adoption and overall higher security.

**Privacy** It is important for IoT applications to recognize when they are handling sensitive data and manage them in the best interest of all parties involved. Thanks to the General Data Protection Regulation (GDPR), EU citizens now have oversight rights over the collection and management of their personal data. Data leaks can have devastating effects on companies. There are therefore economic and legal

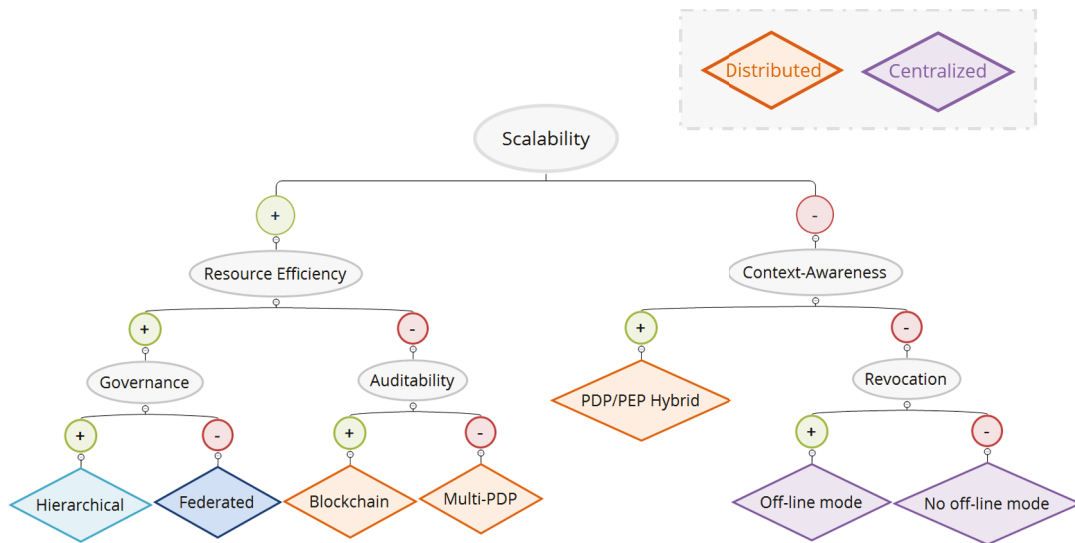


Figure 3.9: A Taxonomy of access control architectures

incentives to put privacy at the center of all IoT applications. This implies transparency about the data collected and its purpose as well as technical means of partially or entirely opting out of it.

**Flexibility** Once deployed, IoT systems are not likely to be heavily modified. The requirements of the system however are likely to evolve over time. Flexibility in the design allows this evolution to happen. Some IoT use cases call for the dynamic addition of users, an important turn over in devices and clients, rapid modifications of policies, collaboration with different entities, etc. Solutions should therefore be built with the expectation and the capacity for change.

**Serverless authorization** There are several reasons why a device might not be connected to its PDP at access time: poor network connectivity, temporary network outage, to limit bandwidth-usage-related costs, to limit power consumption, ... Access control solutions should therefore enable serverless authorization as a temporary or permanent option. PDP can use subject's tokens to deliver updates to their devices, including revocation information. As of today, no solutions enable explicit revocation with serverless authorization.

**Blockchain-based system** The blockchain is in its infancy. Its venture outside of the realm of cryptocurrency is even more recent. However, it is a promising approach to decentralization that should be further explored.

### 3.9 Conclusion

This chapter proposes four architectures for access control - centralized, hierarchical, federated and distributed - and discusses the benefits and disadvantages of each architecture type in the perspective of access control for IoT environments. We provide an extensive review of the literature, with a deep and fine-grained analysis based on detailed criteria for comparing access control solutions against one another. This is the basis for our taxonomy.

From this analysis, we note that only few solutions adopt the hierarchical or federated architectures despite their relevance for IoT use cases. Revocation mechanisms are largely not discussed. Access logs or other means for recording the system activity are barely mentioned outside of blockchain-based solutions. Delegation of privileges is largely unsupported.

In conclusion, though many great solutions have been presented, none solve all the challenges presented by the IoT. We believe future research should explore hybrid solutions. Federation in particular is a requirement for many IoT use cases. Usability is paramount to the adoption of security solutions for consumer facing products. So is privacy. Clearer revocation mechanisms are necessary. Serverless authorization should be enabled to accommodate use cases with low or no network connectivity.

Chapters 4 and 5 present our own access control proposals. Chapters 4 addresses usability and serverless authorization, while Chapter 5 focuses on flexibility and auditability.

# Chapter 4

## OATL: Offline Authorization Token Libraries

### Contents

---

<b>4.1</b>	<b>Introduction</b>	<b>106</b>
4.1.1	Motivation	106
4.1.2	Requirements	107
<b>4.2</b>	<b>Architecture</b>	<b>108</b>
4.2.1	Actors	109
4.2.2	Libraries	110
4.2.3	Access Control Scenario	111
<b>4.3</b>	<b>Tokens</b>	<b>112</b>
4.3.1	Token #1: constrained devices	113
4.3.2	Token #2: privacy preserving	116
<b>4.4</b>	<b>Access Control Process</b>	<b>117</b>
4.4.1	Bootstrap	117
4.4.2	Token request	118
4.4.3	Access request	118
<b>4.5</b>	<b>Security analysis</b>	<b>120</b>
4.5.1	Security assumptions	120
4.5.2	Profiles	121
4.5.3	Assets	122
4.5.3.1	Device-related assets	123
4.5.3.2	General system assets	124
4.5.3.3	Token #1-related assets	124
4.5.3.4	Token #2-related assets	125
4.5.4	Attacks	126

4.5.4.1	Denial of Service (DoS)	126
4.5.4.2	Man-in-the-middle attack (MitM)	126
4.5.4.3	Replay attacks	127
4.5.4.4	Eavesdropping	127
4.5.5	Threats and mitigation	127
4.5.5.1	Denial of Service (DoS)	127
4.5.5.2	Man-in-the-middle attack (MitM)	129
4.5.5.3	Replay attacks	130
4.5.5.4	Eavesdropping	131
4.5.6	Other security consideration	132
<b>4.6</b>	<b>Proof of Concept (PoC)</b>	<b>133</b>
<b>4.7</b>	<b>Comparison to the state of the art</b>	<b>134</b>
<b>4.8</b>	<b>Conclusion</b>	<b>136</b>

---

## 4.1 Introduction

This chapter presents the results that are the most geared towards industrial applications. This is the result of Gemalto's involvement in this doctoral project. Gemalto is an international public company dedicated to digital security with major R&D centers in France, Singapore, and the US. Targeting transparent security, the objective of Gemalto is to design the technology that allows businesses and governments all around the world to offer secure and trusted digital services to their customers or citizens.

Gemalto develops secure software and operating systems embedded in SIM cards, banking cards, tokens, electronic passports, or ID cards which are provided to its clients. The company also deploys platforms and services to manage and protect all the sensitive data these devices hold throughout their life-cycle.

### 4.1.1 Motivation

Chapter 3 analyzes the state of the art in academia. The solutions proposed in the literature are yet to reach industrial adoption. In the industry, access control is often equated to the authentication mechanisms embedded in IoT protocols.

The other popular option is to use IoT platforms run by the giants of the web such as Amazon's AWS and Microsoft's Azure. These platforms are mostly built for sensors that send their data to the cloud where classical access control solutions can be deployed. As a result, all access requests must transit through the server. Modifications to actuators' state are enacted through the server as well. Devices must be connected to the server to receive updates. This centralized approach, in addition to scalability issues, causes policies to lack context-awareness.

The solutions at our disposal are therefore lacking in three ways: actuators are treated as an afterthought, serverless authorization is not an available option, and device-level context is not part of the authorization decision. As a security solution provider, Gemalto's goal is to simplify the implementation



of security measures for its clients. The OATL project sets out to provide a set of token libraries that would handle most of the access control burden and facilitate the work of IoT product developers. This project was realized in conjunction with five of my Gemalto colleagues that participated in the implementation and deployment of a Proof of Concept, as well as the realization of a video introducing the token libraries to developers.

OATL has two goals: provide a serverless context-aware actuator-minded access control solution for users, but also provide flexible pre-packaged access control functions to IoT vendors and developers. The access control layer must therefore be abstracted and isolated. The use of libraries is central to our approach as it fulfills this requirement. This abstraction increases the usability of the solution for IoT vendors as well as the odds of integration in future IoT projects.

Our libraries handle the emission, verification, and client-side management of authorization tokens respectively. Tokens constitute a flexible expression of permissions. Because their interpretation is handled by a library, token formats can evolve independently of the application code. This evolution can stem from evolving cryptographic standards, changes in regulation, etc. Several token formats can even be concurrently emitted by a single server depending on targeted IoT devices.

Token verification is made up of two sub-process: first, the library checks the token's validity (e.g. integrity, not expired, etc), then it verifies that the token's bearer is indeed its intended recipient. The adequacy between the permissions contained in a token and the access request is not checked by the library.

Our libraries can be used by a single business entity developing an IoT device, its companion application, and running a server to issue the token. Several entities could also adopt them to ensure interoperability between the different actors. Authorization Service Providers in particular could handle the issuance of tokens, and point their clients to the libraries they need to verify and manage them inside their applications.

### 4.1.2 Requirements

Let us consider the following use case: a company renting connected cars. Access to each vehicle is controlled by smart locks. Cars can be booked online. Clients only need to download the company app and load their virtual key. No physical contact with a clerk is required. The key should function even when the car is in the middle of the countryside with poor network connectivity. Additionally, the virtual key controls access to the optional car features, depending on the options the client subscribed to. The GPS for instance, is disabled by default. At the end of the lease, mileage information can be extracted for accurate billing. The client should not be able to use the car after that. A client should not be able to rent a car that requires maintenance operations. These cars can be rented for a really short time (to go grocery shopping for instance), or for longer periods (over the weekend).

For such a use case, we need an access control solution that supports:

- **Resource efficiency** - can operate with limited resources,
- **Serverless authorization** - can be used without server connection,
- **Actuator-compatible** - is compatible with both actuators and sensors,

- **Revocation** - enables both implicit and explicit revocation,
- **Granularity** - supports fine-grained authorization,
- **Context-awareness** - can account for context.

This covers the first goal of the OATL project. On top of the requirements defined above, as we are building a solution not only for the users but also for the car renting company and other IoT vendors or service providers, our solution must have the following properties:

- **Ease of Integration** - The integration process should be intuitive,
- **Generality** - The solution should work across application domains,
- **Modularity** - The solution should work with different types of device, protocols, access control models or policies, etc.

To achieve these requirements, OATL proposes a token-based solution. Different types of tokens can be used depending on the restrictions of each IoT device. Three libraries are provided to help developers handle the authorization tokens: a token generation library, a client-side library, and a token interpretation library.

**Contributions** The contributions of this chapter can be summarized as follows:

- A set of libraries that issue, store, and verify tokens, including the legitimacy of the requestor,
- A thorough security analysis of the proposal,
- A Proof of Concept implementation illustrating the, high usability of our solutions for developers and users alike,
- A live coding video demonstration showcasing the ease of integration of our libraries,
- An overall generic access control system enabling serverless authorization.

**Organization** Table 4.1 summarizes the notations used in this chapter.

The rest of this chapter is organized as follows. Section 4.2 introduces the architecture of the solution. Section 4.3 details the different types of tokens available. The access control process is described in Section 4.4. Section 4.5 analyzes the security of the proposed scheme. Section 4.6 presents the Proof of Concept implementation. Section 4.7 compares this solution to the state of the art presented in Chapter 3. Finally, Section 4.8 concludes this chapter.

## 4.2 Architecture

In this section, we present the business and technical actors that play a part in the access control solution. The role of each of our libraries is detailed and an overview of the architecture is provided.

Symbol	Description
$C_D$	Counter number on device
$C_D^{new}$	New counter number on device
$C_t$	Counter number in token
$w$	window size
$n_t$	maximum number of uses for token t
$A_i$	Assumption $i$
$P_i$	Attacker profile $i$
$T_i^{P_j}$	Threat $i$ associated to profile $P_j$

Table 4.1: Notations for OATL

### 4.2.1 Actors

Actors can be separated in two categories: business and technical. We consider two to three business actors: The IoT Device Owner/Vendor, the user, and, optionally, the Authorization Service Provider (ASP). The technical actors are the IoT device (connected car per our example use case), a mobile phone that belongs to the user, the IoT cloud platform deployed by the device owner, and, optionally, the authorization server.

We propose a PDP/PEP hybrid architecture (see Section 3.7.1).

**Device Owner or Device Vendor** There are two cases here. In the first, the device owner is a company. We consider a walled garden ecosystem, meaning that the device owner operates the cloud platform and controls the development of the mobile application. This is a classic scenario in Industrial IoT. It is the case of our car rental use case, but also of smart buildings, or at-home health monitoring kits for instance. The Device Owner is also its administrator. It defines the access control policies and acts as a PAP.

In the second case, the Device Owner is an individual. As part of the product, the Device Vendor operates a cloud platform and provides an application. The application is used to interact with the device. The Owner creates an account on the cloud platform via the application and uses it to manage their device, including defining conditions for its access, adding users, etc. Here, again, the Device Vendor is in control of the mobile app, the cloud platform, and, to an extent, the code running on the device. The Owner acts as a PAP and defines access control policies.

We can illustrate the second case with a smart lock installed in a house. The owner of the house wishes to issue keys for their neighbour to come and feed their cats as they are in vacation, to give to a friend that is staying over while in town, or to a cleaner but only during the time they are scheduled to work at the house. The owner of the house owns the device, the vendor developed the companion app, information are backed onto the cloud, including access control policies.

In the reminder of this Chapter, for the sake of simplicity, we will consider that we are in the first use case. Device Owner will be used to refer to both the Device Owner of the first use case, and the Device Vendor of the second.

**User** The user can be a customer, a patient, an employee, etc. They interact with the device but do not own it and must follow the rules established by the device owner. The user is the subject of the authorization.

**IoT cloud platform** The cloud platform acts as a PDP. It receives access requests, assesses them, and, when appropriate, delivers a token to the user. That token can include additional conditions to be verified by the device at the time of access.

The platform is operated by the device owner. Optionally, the cloud platform can outsource the authorization decision to an authorization server.

**Mobile application** The mobile application is developed under the control of the device owner. Its role is to request an authorization token, store it, and provide it when making the access request. It acts as the user's agent.

**IoT device** IoT devices, behaving either as sensors or actuators, decide whether a requestor is permitted to access some resources (e.g. sensed data), or to perform an action. They do so by retrieving the authorization embedded in the authorization token and, if needed, evaluate the local conditions.

#### 4.2.2 Libraries

OATL is made out of three libraries that are to be hosted on each of the technical actors, namely the cloud platform, the user mobile device, and the device itself. Figure 4.1 illustrates how each library interfaces with the application code developed by the Device Owner. The arrows represent the communication channels required between actors after the initial bootstrap.

The role of each library is described below.

**Token Generation Library (TGL)** The TGL is hosted on the cloud platform. As indicated by its name, it generates tokens based on authorization decisions taken by the cloud platform. The TGL *is not* responsible for the authorization decision. This ensures that the library can be integrated regardless of the authorization engine used by the device owner. The TGL issues different types of tokens depending on the target IoT device.

**Client-side library (CSL)** The CSL is used by the mobile application running on the user's mobile. It has two functions: safely store and manage authorization tokens, and compute the response to device challenges using the information from said tokens.

Here again, the CSL *does not* send the token request to the cloud server, nor does it send the access request to the device, as to not constrain request formats.

**Token interpretation library (TIL)** The TIL is deployed on the IoT Device. Its purpose is to verify each token authenticity and validity. This verification is based on criteria that have nothing to do with the

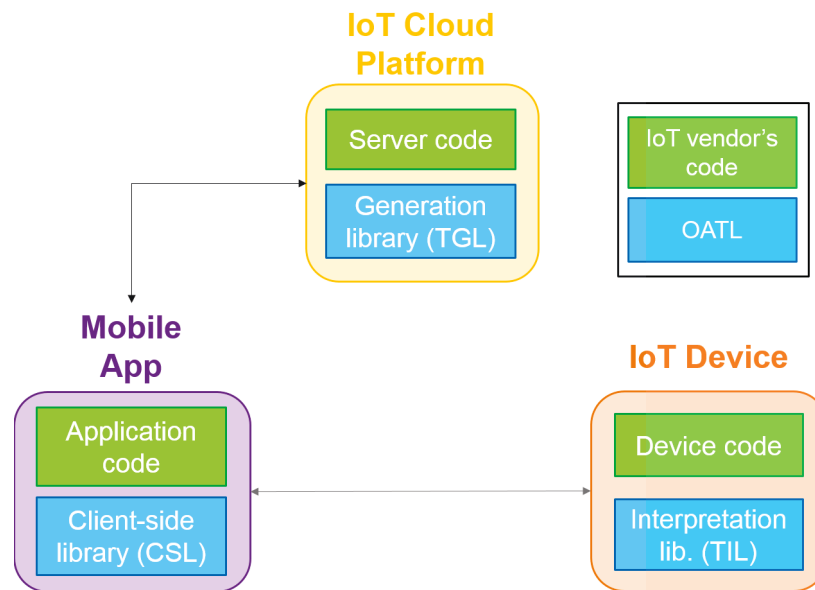


Figure 4.1: OATL architecture

application such as the token's expiration time or its issuer. It generates a challenge message to check that the client that originated the request is the rightful owner of the token.

When a token is deemed valid, the TIL extracts the authorization decision contained within and passes it to the vendor's code to be interpreted.

### 4.2.3 Access Control Scenario

Our access control scenario is composed of two processes: the token request, and the access request. Figure 4.2 illustrates both, taking the car rental use case as an example. This overview excludes the bootstrap phase.

First, the customer books a car using the cloud platform (step 1). The platform processes the booking (including the dates, the payment, etc) and validates it (step 2). This triggers a call to the TGL that issues a virtual key or token embedding the corresponding privileges (step 3). This token is delivered to the application on the user's mobile phone (step 4). This delivery must take place over a secure channel. If the reservation is made directly through the app, the token is part of the booking confirmation. Otherwise, the client can retrieve it later by entering a booking number into the app. Once the token has been delivered, the CSL stores it on the device (step 5). This ends the token request phase.

The client is now next to the car and initiates the access request (step 6). This request must contain the token delivered by the cloud platform. It is retrieved by the CSL. Upon reception, the token's validity is checked by the TIL (step 7). If it passes the initial verification, the TIL generates a challenge message (step 8). The CSL generates the response to this challenge. If the response is correct, the TIL extracts the permissions from within the token and passes them to the device that takes the final authorization

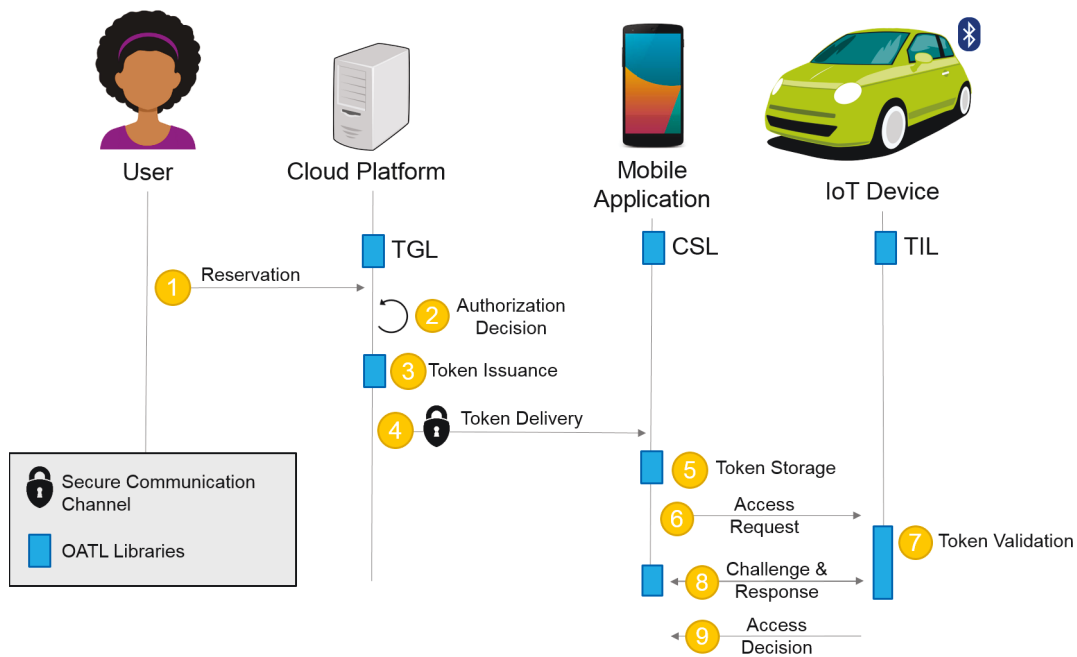


Figure 4.2: OATL example use case - Renting a car

decision (step 9).

### 4.3 Tokens

Tokens are made up of two parts, one for the client and one for the device. They both are given to the client upon a successful token request. The former is for its own use and should not be shared. It contains information used for authentication such as private keys. The latter is meant for the device and materializes the authorization decision. Because of its sensitive nature, the token must be delivered using a secure communication channel. The establishment of such a channel is out of scope.

We propose two formats of token described below. The first one, presented in Section 4.3.1, is geared toward constrained devices. The second one is the object of Section 4.3.2 and focuses on privacy. By design, our solution can be expanded to add other token types. Table 4.2 summarizes the key aspects of each token type.

Tokens are encoded in JSON. This format was also chosen by Hernandez-Ramos et al. [Hernández-Ramos et al., 2013], Bernabe et al. [Bernabe et al., 2016], and Hussein et al. [Hussein et al., 2017b] in their proposals. It is handled by a majority of the programming languages. As such, it is highly interoperable. This choice does not restrain either ours or the Device Owner options implementation-wise.

JSON is not too verbose but still human-readable. This would let actors develop their own libraries to replace either TGL, CSL, or TIL if they are not to their liking without having to look at the original

specifications. It also enable extensions as the syntax is highly flexible and can accommodate the addition of custom fields.

JSON Web Tokens (JWT [Jones et al., 2015b]) are a standardized data format. Along with JSON Web Encryption (JWE [Jones and Hildebrand, 2015]) and JSON Web Signatures (JWS [Jones et al., 2015a]), they are commonly used in the industry.

	Token #1	Token #2
Cryptography	Pre-shared key	Public/Private key pair
Credentials	One key per token (derived from PSK)	One pair per token
Token authenticity	HMAC	Signed by issuer
Authentication	Challenge	Challenge
Time Awareness	No	Yes
Implicit Revocation	Number of use	Expiration time
Explicit Revocation	Validity window update	TBD

Table 4.2: Token types

#### 4.3.1 Token #1: constrained devices

This type of token uses symmetric cryptography and is built to work with device without time-awareness. A counter is used instead. A type #1 token contains the following fields:

- **Token Identifier** - *String*. Uniquely identifies the token.
- **Token Type** - *String*. Indicates the token type (1 or 2).
- **Token Key Parameters** - *String*. Parameters used to derive the token key
- **Token counter number** - *Integer*. Indicates the number of tokens issued by the TGL so far.
- **Issuer** - *String*. Optional. ID of the cloud platform
- **HMAC** - *String*. Used to ensure the authenticity and integrity of the token
- **Update window size** - *Integer*. Disregarded is negative. Updates the size of the validity window otherwise.
- **Max number of use** - *Integer*. Optional. Used for implicit revocation. Indicates how many times the token can be used before becoming invalid. If not included, the token is single use.
- **Authorization Object** - *JSON*. Permissions materialized by the token. Contains the following fields:

- *Object* - *String*. Object of the authorization
- *Actions* - *String array*. Actions allowed on the object
- *Conditions* - *JSON*. Optional. Local conditions to be checked at access time.
- *Subject* - *String*. Optional. Id of the client

Signatures and keys are encoded in Base64.

**Key derivation** A type #1 token involves five types of keys, all of which are symmetric. They are represented on Figure 4.3. The first key is a HMAC key that is shared by the device and the cloud platform. It is used for integrity and authenticity. The second is a master Pre-Shared Key (PSK) that is also shared during the bootstrap phase. The three remaining types of keys are derived from this PSK. They are the Token Key, the Challenge Key, and the Session Key.

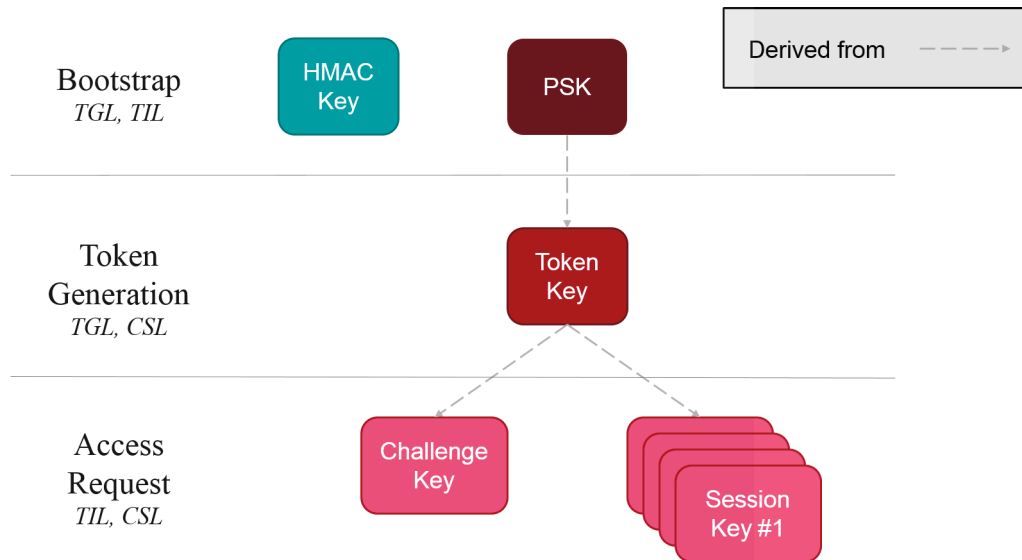


Figure 4.3: Token #1 - Keys

When a token is issued, the TGL derives the Token Key from the PSK. This key is included in the client-side token. The derivation information are included in the device-side token. A different Token Key is used for every token. This enables subject unlinkability across tokens, provided the subject ID is not included in the authorization object.

When it receives the token, the CSL uses the Token Key to generate a Challenge Key. The generation information will be included with the access request. This derivation only occurs once. The same Challenge Key will be used throughout the token life.

After the access request has been validated, the Device and Mobile app exchange the information used to derive the session key from the Token Key. A different Key is used for each session.



**Revocation** There are two types of revocation to consider: implicit and explicit revocation. The former is embedded in the token and limits its use over time, the latter requires a trigger.

**Implicit Revocation** Without time awareness, type #1 tokens rely on a counter instead [Wu et al., 2017]. This counter must be synchronized between the TGL and the TIL. The TGL increments it each time a token is issued. The TIL increments it depending on the tokens that it sees. Additionally, the TIL must be configured with a window size. This defines the number of tokens that can be valid concurrently. This parameter can be modified by the TGL to adapt to the number of issued tokens or to implement explicit revocation.

Let  $C_D$  be the current counter number on the device,  $C_t$  be the counter number of token  $t$ ,  $n_t$  be the maximum number of uses for token  $t$ , and  $w$  be the window size.

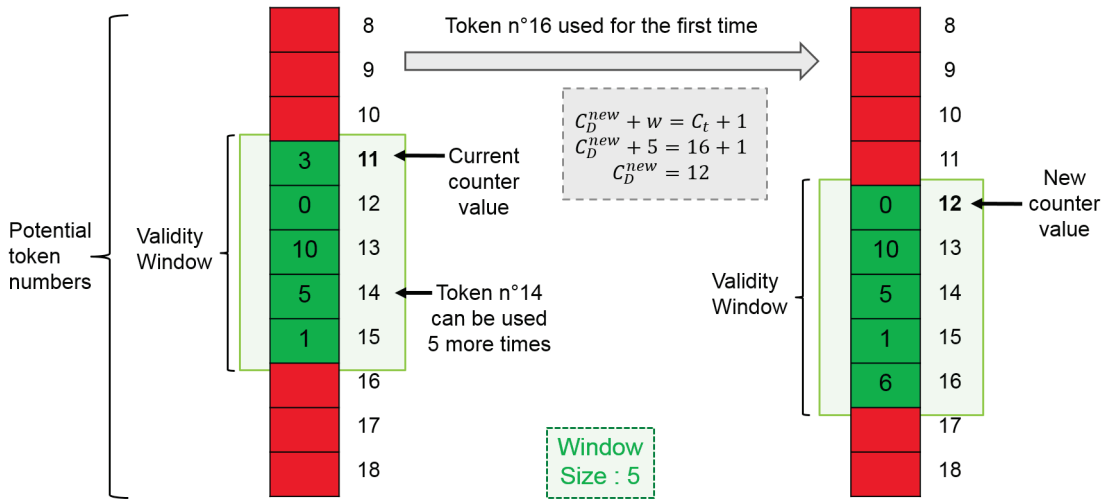


Figure 4.4: Implicit revocation without time-awareness

When presented with a token for the first time, the device compares its own counter number  $C_D$  to  $C_t$ . If  $C_t < C_D$ , the token falls outside of the validity window and access is denied. If  $C_D \leq C_t < C_D + w$ , the remaining number of uses for this token is set to  $n_t - 1$  and access is granted. If  $C_D + w \leq C_t$ , the window slides to meet the token:  $C_D$  is updated such that  $C_D^{new} + w = C_t + 1$ .

After that initial connection, the token is valid if  $C_D \leq C_t < C_D + w$  and the number of remaining uses is not null.

Figure 4.4 illustrates this process. Simply put, access is only allowed up to  $n_t$  times as long as the counter number of the token  $C_t$  is within the acceptable window. In the example, token 14 can be used five more times, whereas token 15 can only be used once. The valid token counter number are in green, invalid in red.

When a token with counter number greater than the acceptable window is presented, the counter on the device is updated to slide the window forward accordingly. So when token 16 is presented, the window slides to adjust and token 11 becomes invalid, even though there were 3 uses remaining.

The window never slides backward. This means that tokens may become unusable before their allotted number of uses. The window can be configured to fit specific use cases and depends on the device's memory and the number of expected clients.

**Explicit Revocation** Explicit revocation is achieved by adjusting the size of the validity window as illustrated on Figure 4.5. It is not really precise however and will revoke all tokens issued up to that point.

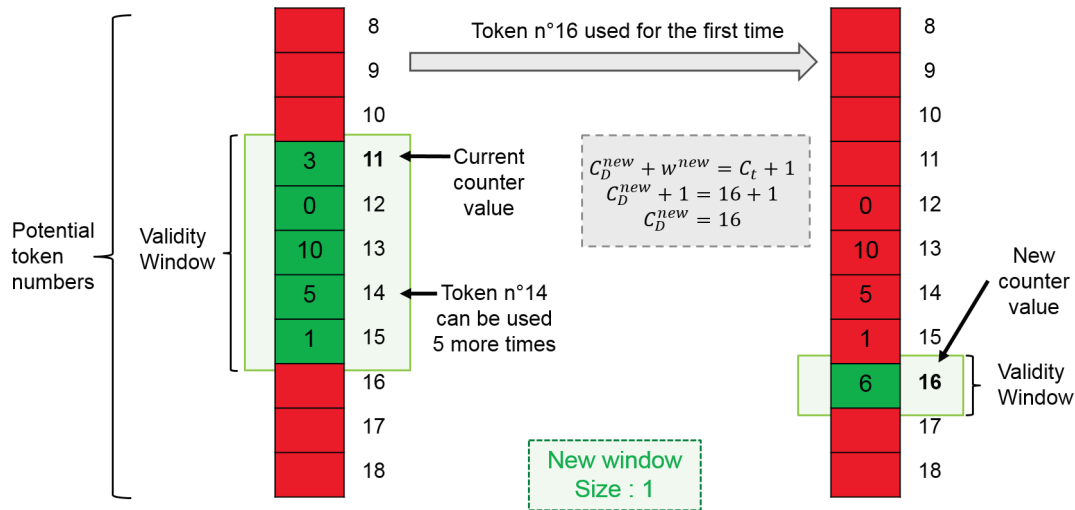


Figure 4.5: Explicit revocation without time-awareness

To do so, the TGL issues a token setting the window size at 1. This will cause the  $C_D$  to slide forward to this  $C_t$ , thus making every older token invalid. In our example, when token 16 is presented, the  $C_D$  is set to 16 and all other tokens are revoked. If the new window size had been 2, 3 or 4 instead of 1, only the 3, 2, or 1 oldest tokens would have been revoked respectively. The size of the window can be increased again later to re-align with the use case.

Revocation does not require a connection to the server. It may require the intervention of a trusted user in the case of explicit revocation. In our car rental use case, a company employee could fill this role. Another solution is to embed it in the process: when a customer *checks out*, a new token is issued to allow access to the mileage information. This token can be used to revoke older permissions.

### 4.3.2 Token #2: privacy preserving

This type of token uses asymmetric cryptography and is built to work with time-awareness. A type #2 token contains the following fields:

- **Token Identifier** - *String*. Uniquely identifies the token

- **Token Type** - *String*. Indicates the token type (1 or 2)
- **Token's public key** - *String*. Short term credentials linked to the token
- **Issuer** - *String*. Optional. ID of the cloud platform
- **Issuer's signature** - *String*. Used to ensure the authenticity and integrity of the token
- **Issued at** - . Time of token issuance
- **Valid From** - . Time at which the token becomes valid
- **Expires at** - . Time after which the token is revoked
- **Authorization Object** - *JSON Object*. Permissions materialized by the token. Contains the following fields:
  - *Object* - *String*. Object of the authorization
  - *Actions* - *String array*. Actions allowed on the object
  - *Conditions* - *JSON Object*. Optional. Local conditions to be checked at access time.
  - *Subject* - *String*. Optional. ID of the client

Additionally, the client-side token contains the private key associated to the token's public key. ECDSA is used for signatures. A different key pair is generated for each token to ensure client unlinkability across token.

Implicit revocation is handled by the expiration time included in the token. Explicit revocation is not yet handled for this token.

## 4.4 Access Control Process

### 4.4.1 Bootstrap

Here we define the information that must be exchanged between the different actors before the access control process can begin. Discovery is out of scope.

**Device Registration** To make its resources available, a device must associate with the cloud platform. During this bootstrap phase, the device will describe its resources and choose the kind of token it will work with.

The device owner registers new devices on the platform, including the type(s) of token they support. A device can support both token types. A unique identifier is generated and associated with the device.

For type #1 tokens, the device and the platform exchange a PSK and a HMAC key. They also synchronize their counters. For type #2 tokens, the platform provides its certificate to be stored in the device. The device will need it at access time to ensure the token's authenticity.

Finally, the device owner declares a list of resources within the device and their corresponding access control policies. This final step completes a device's registration.

**Client Registration** Where the libraries are concerned, no information need to be exchanged between client and platform before access. On the application level however, a registration phase likely occurs before the authorization token can be delivered. In our example use case, client registration is part of the booking process.

The client may specify its preferred token type. The preference is not guaranteed to be respected as device requirements will always prevail. It should however be respected whenever possible.

#### 4.4.2 Token request

After the bootstrap is complete, the client places a token request with the cloud platform. This original exchange is likely to involve an authentication between the two actors. The format of this request is out of the scope of our proposal. The communication protocol between client and cloud platform is out of the scope of our proposal. This choice was made to enable the integration of our libraries into existing environment. Because our main goal is usability, we do not want to hinder the developers by adding unnecessary architectural requirement. However, we recommend TCP/IP for this segment of communication.

For similar reason, the libraries do not handle the authorization engine. Our libraries can therefore be integrated into any policy or any model, provided it supports CapBAC as an access control mechanism. The authorization process can also be offloaded to an authorization service provider.

After the authorization decision has been taken, the corresponding Authorization Object is created. It must contain the name of the resource and the actions allowed on the object. Optionally it might contain either the name of the subject (or a pseudonym), or local conditions to be verified by the device at access time.

These local conditions must be expressed using the JSON format. They are not evaluated by our libraries and simply passed to the vendor's code instead. This gives the vendor the freedom to express conditions however they want rather than be forced into a specific format.

The Authorization Object is passed to the TGL along with the ID of the device for which this token is to be issued. Based on that information, the TGL selects the appropriate format, derives or generates the appropriate key(s), signs, and returns the token to the vendor's code. Here again, communication is not handled by the TGL. The cloud platform can then either return the token to the client right away or store it to be retrieved at a later date.

The token must be sent over a secure channel. It is received by the mobile application that invokes the CSL. The CSL separates the token into its client and token side. The client side is stored either in an hardware or software secure storage, depending on the underlying phone. Both of the token sides are mapped to an application-provided identifier to be retrieved for the access request. The application may potentially notify to the server that the token has been received and successfully processed.

#### 4.4.3 Access request

Figure 4.6 illustrates the access request flow. It involves the mobile and the CSL as well as the device and the TIL. The access request does not involve the cloud platform and can be operated offline. Once again, we do not specify the communication protocol used between the client and the device. However,

we recommend a short range protocol such as BlueTooth Low Energy (BTLE). Discovery mechanisms between the two actors are out of scope (step 1). The format of the access request (step 3) is left to the vendor. The only requirement is that the request contains the device-side token, as delivered during the token request. This token is recovered by invoking the CSL (step 2). In the case of type #1 tokens, the request also includes the information used to derive the Challenge Key. They are not included in the token as they are generated by the CSL.

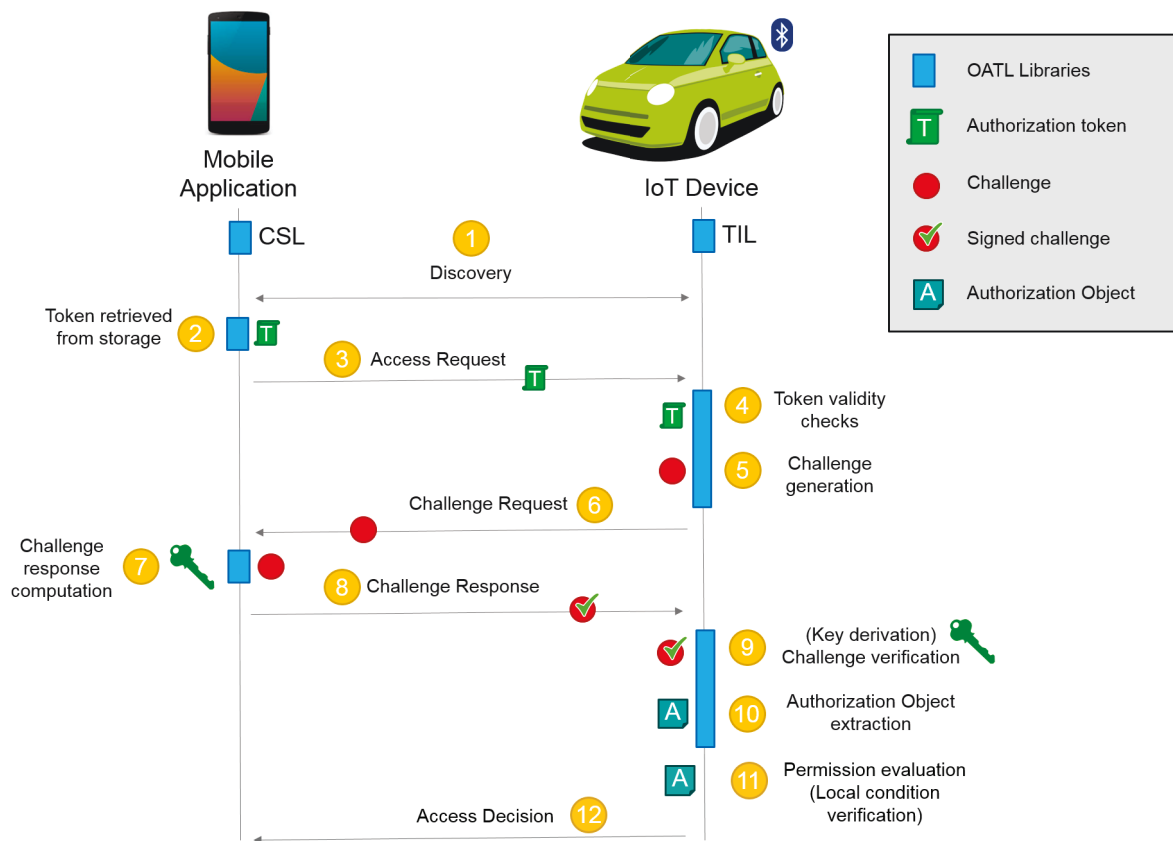


Figure 4.6: Access Request

Upon reception, the vendor's code running on the device invokes the TIL to check the token's validity (step 4). First, the TIL checks whether the token falls within the validity window (type #1 token) or if it is expired (type #2 token). If it passes this test, the TIL checks the token's integrity and authenticity, either via HMAC (type #1 token) or by verifying the cloud platform's signature (type #2 token). On a successful check, the TIL generates a challenge message and returns it to the vendor's code (step 5). Otherwise, an empty string is returned. The challenge message is sent to the mobile application (step 6)

The application invokes the CSL to generate the response to the challenge (step 7). The mobile application then sends it back to the device that passes it to the TIL (step 8). At that point, if appropriate, the TIL generates both the Token Key using the information contained in the token and the Challenge Key

using the information contained in the request. The TIL uses the appropriate key to verify the challenge (step 9). If it passes, the window size and remaining number of uses are updated. The Authorization Object is returned to the vendor's code (step 10).

At that point, the token is deemed valid. However, the adequacy between the request and the permissions contained within has not been verified. This is a job for the vendor's code (step 11). The request can therefore still fail at this point. If it does and in the case of type #1 tokens, the token is still considered as having been used once. The local conditions are also verified outside of the TIL (still step 11). For performance optimization and bandwidth efficiency, these checks should occur before the challenge message is sent out. However, from the developer's point of view, it makes more sense to only intervene after the job of the TIL is finished rather than in the middle. Finally, the access decision is returned to the mobile application (step 12).

## 4.5 Security analysis

In this section we analyze the security of our proposal. Our analysis revolves around the token request and the access request. First, we lay out our security assumptions. Second, we propose five attacker profiles along with their motivations and capabilities. Third, we detail our assets and their sensitivity levels. Finally, we examine potential threats and how our solution addresses them.

### 4.5.1 Security assumptions

We work under following security assumptions:

- A1 **Secure cryptographic primitives** - The underlying cryptographic primitives are assumed to be secure.
- A2 **[Strong] Robust random generator** - We assume devices have access to a secure source of randomness.
- A3 **Secure communication channel (token request)** - A secure communication channel can be established between the cloud platform and the mobile. This channel is used to exchange the token after a successful token request.
- A4 **[Strong] Secure libraries** - We assume our libraries cannot be compromised. The code running on top however, can be modified.
- A5 **[Strong] Compliant cloud platform** - Even when the cloud platform is not operated by the same actor as the device, it is assumed that tokens issued by the cloud platform are in compliance with the policies defined by the device owner or administrator. Furthermore, the cloud platform must provide usable tokens to the users and cannot actively work to disrupt operation.
- A6 **[Strong] Secure execution environment (cloud server and device)** - We assume that the code running on the cloud server and IoT device is executed in a secure environment.

- A7 **[Strong] User authentication (on mobile)** - We assume that the mobile application requires the user to authenticate locally before using the private key. Consequently, if someone were to steal the mobile phone, they should not be able to use the application.
- A8 **Secure storage on mobile** - We assume that the mobile phone has a secure storage that can be used by the CSL.
- A9 **[Strong] Authentication between user and cloud platform** - We assume that the cloud platform is able to authenticate a user before delivering a token. The authentication methods is out of scope.

#### 4.5.2 Profiles

In the following we assume there are multiple instances of each actor in the system. This means that a device may be registered with several cloud platforms, a cloud platform serves multiple devices, clients can be registered with multiple platforms and seek access to multiple devices. Different instances may be operated by different actors that do not trust each other.

Despite this multiplicity and for the sake of simplicity, we restrain the scope of attacks to a single actor: the cloud platform will try to spy on the user, the user will try to access resources on the device, ...

We consider five types of attacker profiles, numbered from  $P_1$  to  $P_5$ . For each, we detail their capabilities and goals.

**P1 - Malicious cloud platform operator** According to assumption A5, cloud platforms can be trusted to comply with access control policies when issuing tokens. Their operator can however be curious or malicious with regards to resources and users that are not registered with them. A cloud platform operator may wish to learn information about user's usage patterns, or their possible registration with other cloud platforms. They may also be interested in learning about resources a device owns but may have registered with another platform, or devices that are not registered with them at all. This boils down to a breach in privacy. Additionally, a cloud platform operator may try to issue fraudulent tokens for resources or devices that are not registered with them. They can also try to use tokens issued for legitimate users in their stead.

Depending on the type of token used, the cloud platform may share symmetric keys with the device. It holds information on a device's resources and on clients. The nature of this information is unclear as we do not control the way policies are expressed, nor do we have control over the authentication and authorization process for clients. Clients contact the cloud platform when new tokens are required. The cloud platform therefore knows when a user is able to gain access to a resource, though it does not know when an access request is placed. Most importantly, the cloud platform has access to token keys generated for clients.

**P2 - Malicious device owner** We consider a curious device owner that wants to gain information about the user without their knowledge. The device owner is also interested in gaining information about other devices, potentially via a common user. The owner may use the information transiting through their device to try and access other devices, maybe by pretending to be the common user.

As the owner, *P2* controls the code running on the device. They also have access to the tokens presented by users. Depending on the type of tokens used, the device owner also has access to a number of symmetric keys shared with the cloud platform.

**P3 - Malicious user** Malicious users wish to gain access to resources they are not allowed to access. They may also wish to learn information about other users or map the resources available on a given device in addition to those they have access to.

The malicious users control the mobile phone that runs the application. They can therefore change the code running on top of the CSL library. They have access to the information contained in the token.

**P4 - Malicious third party** The purpose of the malicious third party is to gain as much information as possible. This means learn who the users are, when they access each device, what they request, whether they are successful, what are the resources hosted on each device, what cloud platform protects them, how they can be accessed. When the device is an actuator, the malicious third party is also trying to operate said device. The malicious third party might want to disrupt operations, or compromise the technical actors.

We consider that the malicious third party can eavesdrop on the different exchanges that happen over unprotected channels as well as intercept messages and send their own.

**P5 - Colluding cloud platform and device** Here we consider that the cloud platform is colluding with the IoT device. As such, *P5* enjoys the capacities and knowledge of both the cloud platform and IoT device as described above. The purpose of this collusion is to lift the pseudonymity of the user and determine a precise usage pattern. This collusion corresponds to our example use case where the car rental company also operates the cloud platform, acting as both the platform operator and the device owner.

### 4.5.3 Assets

Figure 4.7 presents our assets. Here we consider that we are working with a single cloud platform, several devices, and several users. Colors encode sensitivity levels. **Black** denotes a system-wide critical asset. If such an asset is compromise, the system cannot operate (i.e. devices and their functions cannot be accessed) or the access control process cannot be trusted (e.g. an attacker has gained the ability to counterfeit tokens). **Red** denotes a user-wise or devise-wise critical asset. If such an asset is compromised, a few users can no longer access a device or can be impersonated. Other users can still use the system. Likewise a device might become unreachable while other devices continues to operate as normal. **Orange** corresponds to limited functionalities. If such an asset is compromised, a user might need to renew their token, privacy might be compromised, etc. **Green** assets represent metadata leaks not easily linkable to a given user or device. **Blue** is out of scope.

Assets are of two types: assets introduced by our access control system and assets that are specific to the application itself. For instance, the data stored on the device are application specific whereas the privileges afforded to a user are an asset generated by our access control system. Among these latter



Assets		Confidentiality	Integrity	Availability
Device	State	Limiting	User/Device critical	User/Device critical
	Data	Limiting	User/Device critical	User/Device critical
	Identity	Limiting	Acceptable	Acceptable
General	User Identity	Limiting	Acceptable	Acceptable
	User privileges	Limiting	User/Device critical	Acceptable
	Public token	Limiting	User/Device critical	Limiting
	Private token	User/Device critical	Limiting	Limiting
Token 1	Secret keys	PSK	User/Device critical	User/Device critical
		HMAC	User/Device critical	User/Device critical
		Token Key	User/Device critical	Limiting
		Challenge Key	User/Device critical	Limiting
		Session Key	Limiting	Limiting
	Counter Number	Acceptable	User/Device critical	User/Device critical
	Remaining number of use	Acceptable	User/Device critical	User/Device critical
Token 2	Token private key	User/Device critical	Limiting	Limiting
	Cloud platform signing key	System critical	System critical	User/Device critical

	Acceptable		User/Device critical
	Limiting		System critical

Figure 4.7: OATL - Assets sensitivity level

assets, we distinguish between those that are generic, such as the previously mentioned user privileges, and those that are linked to a specific token type.

For each asset, we consider its confidentiality, integrity, and availability and the sensitivity of their compromise.

#### 4.5.3.1 Device-related assets

**Device state** On an actuator, the device state can be changed by calling the appropriate functions. In our car rental use case for instance, the maximum speed of the car, the temperature of the Air Conditioning, or the state of the smart lock are device states. If the **confidentiality** of the state is compromised, we have a privacy concern but no impact on operations. Compromised **integrity** and **availability** prevent the normal course of operation for this single device. However, if an attacker can modify an actuator's state,

the access control system has failed at preventing unauthorized access.

**Device-generated data** Device-generated data are the whole purpose of sensors. They can also be relevant in actuators. In our car rental use case for instance, the GPS position or the car mileage are device-generated data. A breach in **confidentiality** does not prevent legitimate users from accessing the data whereas a breach in **integrity** or **availability** does. However, if an attacker can read sensor data, the access control system has failed at preventing unauthorized access.

**Device identity** As the discovery process between devices and users is out of scope, we only consider the device identity privacy-wise. In that context, if an identity's **confidentiality** is compromised, privacy has been breached.

#### 4.5.3.2 General system assets

**User's identity** User authentication is out of scope. As such, we only consider user identities privacy-wise. In that context, if an identity's **confidentiality** is compromised, privacy has been breached.

**User's privileges** A breach in privilege **confidentiality** compromises user privacy but does not affect operation. Attacking the **integrity** of user privileges can be equated to privilege elevation or suppression. It either prevents a legitimate user from accessing authorized objects or breaks the access control system by enabling unauthorized access to a malicious user.

Our libraries do not cover the authorization engine. As such, the availability of user privileges is out of scope.

**Public token** As its name supposes, the consequences of breaching a public token's **confidentiality** are limited to privacy concerns. In the case of **unavailability**, a new token can be requested. An **integrity** compromise can have consequences ranging from the need for a new token to privilege elevation.

**Private token** Private tokens are more sensitive as they contain the private or secret keys corresponding to the token. Depending on the token, this can lead to the attacker enjoying the privileges of the intended recipient (**confidentiality**). In case of **unavailability** or **integrity** issues, the user can request a new token.

#### 4.5.3.3 Token #1-related assets

**Secret keys** Type #1 tokens involve a number of secret keys. Here we consider that each PSK and HMAC key is used by a unique cloud platform/device pair. That way, if a key is lost or otherwise compromised, the security breach is contained.

If the **confidentiality** of *both* the PSK *and* the HMAC key is concurrently compromised, an attacker can forge any token they like, thus breaking the access control system for the corresponding device.

The HMAC key alone would not be enough as the attacker would be unable to create a valid token key: the TIL computes the token key by deriving the PSK. Creating a valid token key without the PSK would break Assumption A1.

The PSK alone would not be enough: the fraudulent tokens would not pass the authenticity check from the TIL. Creating a valid token key without the HMAC key would break Assumption A1.

Compromising the **integrity** or **availability** of the PSK and HMAC key prevents the cloud platform from issuing new tokens and the device from verifying them, thus stopping operation for the corresponding device.

A breach in either confidentiality or integrity of the PSK and HMAC key affects the system until keys are replaced on both the cloud platform and the IoT device, thus forcing a new configuration phase. Key security can be increased however by using a Hardware Secure Module (HSM) for storage.

A token key with compromised **confidentiality** enables the attacker to enjoy the privileges associated with the token. However, the impact of such compromise is limited by the use of the Challenge key. That key is derived by the mobile application and fixed over the life of the token. This means that the first person to use a token sets the challenge key. An attacker would therefore need to either set it themselves in which case the user would be alerted as they would not be able to use the token. Or they could capture the exchange between the mobile and the device to retrieve the derivation information used to derive the challenge key.

Compromising the **confidentiality** of the challenge key is not enough either as the token key is required to generate valid session keys between mobile and IoT device. So in order for an attacker to impersonate a user, they must possess both the token and the challenge key.

The session key itself is short-lived. If its **confidentiality** were to be compromised, the window of exploitation would be really short. It can still be used to decipher previous communications.

A breach in either **integrity** or **availability** for the token key, the challenge key, or the session key requires the user to request a new token or start a new session which is minor.

**Counter number & remaining number of use** It is difficult to extract meaningful information from the counter number of a device or the remaining number of use for a token. At best, an attacker can know at which rate a device is used. Tokens are not linked to users. So the knowledge that Token 5 has been used three times is worthless on its own. Hence, **confidentiality** is not an issue. Note that crossing that information with the knowledge of a malicious cloud platform breaks pseudonymity.

The **integrity** and **availability** of these parameters however is important. Without them, the device can no longer check token validity, thus halting operation. If an malicious user can modify both, they can use their token indefinitely by fixing the validity window and replenishing their allotted use. With only the counter number, an attacker can move the validity window so far ahead that new tokens are considered expired. They can also move it back to use tokens that are below the validity window with some remaining uses. With only the remaining number of use, a malicious user can use a token more than intended as long as the validity window does not slide away. An attacker can expire all tokens by setting their number of remaining uses at 0. Manipulating these value can be very disruptive. However, one can only gain access to objects that they were once authorized to access.

#### 4.5.3.4 Token #2-related assets

**Token private keys** With a token private key(**confidentiality**), an attacker can use the token instead of its intended recipient. A compromise in **integrity** or **availability** requires only the issuance of a new token

to be fixed.

**Cloud platform signing key** By breaking the **confidentiality** of the cloud platform private key, an attacker can forge any token they like for any device, thus breaking the system entirely. Similarly, if the **integrity** of the key is compromised, tokens can no longer be issued. Both attacks last until the key is revoked and a new public key is deployed across devices, forcing a new system-wide re-configuration. Key security can be increased however by using a Hardware Secure Module (HSM) for storage.

If the signing key is **unavailable** for a short time, the system can continue to function as it is not required at access time. However a prolonged availability issue would halt operations.

#### 4.5.4 Attacks

Let  $\mathcal{A}$  be the attacker. In this section we consider four means by which  $\mathcal{A}$  can try to disturb the system and compromise our assets. They are Denial of Service attacks, Man-in-the-middle attacks, replay attacks, and eavesdropping.

##### 4.5.4.1 Denial of Service (DoS)

In DoS attacks,  $\mathcal{A}$ 's purpose is to block communications or render actors incapable of operating their services. Our proposal involves two communication segments that can be interrupted: Mobile to platform communication during token requests, and mobile to device communication during access requests. Consequently, the actors that would be targeted by such attacks are the cloud platform and the device.

DoS target availability.

##### 4.5.4.2 Man-in-the-middle attack (MitM)

By placing themselves between two actors and intercepting their communications,  $\mathcal{A}$  hopes to gain access to private information or pass as a legitimate user, thus either obtaining an authorization token (MitM during token request), or access to a given resource (MitM during access request).

This type of attacks can occur in a synchronous or asynchronous fashion. The former are called relay attacks while the latter are delayed Man-in-the-middle.

MitM target confidentiality and potentially availability.

**Relay attack** In relay attacks, the attacker artificially bridges the distance between the two legitimate actors by relaying messages over a potentially long distance. This attack is used to fool proximity-based protocol. An attacker could for instance stand by the hotel pool, next to a victim's phone, and relay messages from the phone to an accomplice standing by the smart lock of the hotel room.

**Delayed Man-in-the-Middle** Alternatively, the man-in-the-middle attack can be conducted in two steps. First, the attacker registers a sequence of messages exchanged between them and one of the actors (here the mobile application). Then, the attacker replays the registered sequence to the intended party.

This attack is related to replay attacks. The difference is that the messages are new to the intended recipient. We take the example of a counter-based replay protection mechanism: each time  $A$  and  $B$  exchange messages, they increment a counter. Replayed messages will present an outdated counter number. Whereas delayed MitM presents a fresh number and passes through the protection.

#### 4.5.4.3 Replay attacks

By replaying intercepted messages,  $\mathcal{A}$  hopes to access unauthorized resources. This replay can affect either the token request or the access request.

#### 4.5.4.4 Eavesdropping

$\mathcal{A}$  hopes to gain information by observing communications. The observation is passive. Under this attack we also consider everything that malicious IoT platforms, users, or device owners can achieve with the information they can access through their normal use of the system.

### 4.5.5 Threats and mitigation

We have defined five attacker profiles, four means of attack, and detailed our system assets. We now examine what threats our system is subject to and how we mitigate them. Figure 4.8 summarizes this discussion. In green are the threats that are handled directly by our libraries. In orange, the threats that are either mitigated by the library or for which we have provided a mitigation in the discussion. In gray, the threats that are not addressed by OATL. When no color are provided, the risks for the given attacker to compromise the given asset with the given method are low to null.

#### 4.5.5.1 Denial of Service (DoS)

A collusion between the cloud platform administrator and the device owner ( $P5$ ) has no interest in disturbing availability as either of these actors would be the target in the case of a DoS attack.

**Cloud platform (token request)** The cloud platform acts as a classical server. As such, it can use classic security solutions to mitigate DoS attacks. Our proposal does not control the specifics of token requests or administrative services that might be offered by the cloud platform. As such, we cannot assess the likelihood of a successful attack. For the same reason, we cannot conclude whether a given attacker profile is more likely to mount such an attack or succeed. We can still exclude P1 and P5 as according to  $A5$  the cloud platform cannot disturb its own operation.

	Assets										Attacks									
	DoS					Man-in-the-middle					Replay					Eavesdropping				
Device	P1	P2	P3	P4	P1	P2	P4	P1	P2	P4	P1	P2	P4	P1	P2	P4	P5			
State	C	I	A	C	I	A	C	I	A	C	I	A	C	I	A	C	I	A		
Data	C	I	A	C	I	A	C	I	A	C	I	A	C	I	A	C	I	A		
Identity	C	I	A	C	I	A	C	I	A	C	I	A	C	I	A	C	I	A		
User Identity	C	I	A	C	I	A	C	I	A	C	I	A	C	I	A	C	I	A		
User privileges	C	I	A	C	I	A	C	I	A	C	I	A	C	I	A	C	I	A		
Public token	C	I	A	C	I	A	C	I	A	C	I	A	C	I	A	C	I	A		
Private token	C	I	A	C	I	A	C	I	A	C	I	A	C	I	A	C	I	A		
Token 1	Secret keys	PSK	C	I	A	C	I	A	C	I	A	C	I	A	C	I	A	C	I	A
		HMAC	C	I	A	C	I	A	C	I	A	C	I	A	C	I	A	C	I	A
		Token Key	C	I	A	C	I	A	C	I	A	C	I	A	C	I	A	C	I	A
		Session Key	C	I	A	C	I	A	C	I	A	C	I	A	C	I	A	C	I	A
Token 2	Challenge Key	C	I	A	C	I	A	C	I	A	C	I	A	C	I	A	C	I	A	
		Counter Number	C	I	A	C	I	A	C	I	A	C	I	A	C	I	A	C	I	A
		Remaining number of use	C	I	A	C	I	A	C	I	A	C	I	A	C	I	A	C	I	A
Token 2	Token private key	C	I	A	C	I	A	C	I	A	C	I	A	C	I	A	C	I	A	
		Cloud platform signing key	C	I	A	C	I	A	C	I	A	C	I	A	C	I	A	C	I	A




	Handles		Mitigated		Not addressed	C	Confidentiality	P1 – Cloud Platform
						I	Integrity	P2 – Device Owner
						A	Availability	P3 – User
								P4 – Third Party
								P5 – Collusion

Figure 4.8: Threats addressed by OATL

In the context of our threat model, a DoS targeting the cloud platform would threaten the availability of its PSK and HMAC key, or the signing key for token #1 and #2 respectively. This would prevent the issuance of new authorization tokens. Ongoing operations however would go on as the cloud platform is not required for the access itself. This neutralizes short-lived DoS attacks that would have limited consequences.

**Device (access request)** DoS attacks on the device target the availability of the device's state and data. The device's resource constraints make it particularly sensitive to DoS attacks. The memory usage for instance, can be abused so that the device is no longer able to function. Because of this, challenges carry a time-out and only allow a single concurrent challenge per token. This to avoid a malicious application flooding the device with unfinished access requests.

The window-size for type #1 tokens is used to control how many tokens can be accepted at once. This should be set according to use case specifications and device capabilities. This window can be adjusted over the course of operation to accommodate an increase in user access requests or to mitigate malicious behavior.

Bandwidth-wise, the token validity is checked before a challenge is sent out. This prevents malicious users from artificially increasing bandwidth usage using expired tokens or simply placing requests with no token at all. Computation-wise, type #1 tokens only generate keys after the challenge answer has been received. Keeping the most expensive operations for last.

A token is still marked as used if the permission does not match with the request. This limits the number of attempts a malicious user (*P3*) can place with a single token. The cloud platform should blacklist a user that requests tokens too often, thus preventing them from spamming the device. A user could request tokens often to obtain a high token number and revoke older tokens before they can be used. The same token request monitoring will prevent this behavior.

Communications between the mobile and the device use short range protocols. A protocol-level DoS therefore has limited consequences on the system as a whole.

#### 4.5.5.2 Man-in-the-middle attack (MitM)

Once again, we have no control over the specifics of the token request and it might very well be vulnerable to such an attack. However, *A3* guarantees a secure communication channel for token delivery. This ensures the private token's confidentiality.

Our focus is on the access request. In this context, MitM attacks target the confidentiality of the device state and data, user identity and privileges as well as public tokens as all of these assets are involved in the access request. The access request can take place over an insecure channel, though this breaks the confidentiality of the user privileges. The token only contains a short-lived pseudonym. Hence the user identity is not revealed to a third party observer (*P4*). However, a malicious cloud platform can link the token to a subject and gain unauthorized knowledge of its user's activity.

Once access is granted, the device and mobile should switch to a secure communication channel before proceeding. The session key for this channel is derived from private information possessed by each party. Based on the public information alone, an attacker cannot compute the session key without

breaking  $A1$ . A malicious cloud platform however has access to the same private information as the user. They can therefore compute the session key and keep eavesdropping on communications.

In addition to confidentiality breach, the attacker may also try to modify the intercepted access request. If permissions contained within the token are larger than those needed for the current access request, the attacker can request access to another resource. To limit this attack surface, cloud platform should be mindful of including the smallest possible authorization set in tokens, breaking up permissions between different tokens if need be. This leaks more information about the user's activity to eavesdroppers but ultimately protects access to the device's data and functions.

An attacker cannot modify a token to gain access to resources that are not authorized to the user. This would require changes to the public token which cannot be achieved without breaking the underlying cryptographic primitives (Assumption  $A1$ ). All attempts will be detected by the integrity check performed by the TIL. Because the cloud platform can manufacture token for themselves, they have no incentives to modify the token here.

Finally, an attacker that has successfully planted themselves in-between the mobile application and the device can cut communication between them, thus threatening the availability of the device's data and state. By  $A5$ , the cloud platform cannot engage in such behavior as it would disturb the flow of operations.

The mobile application ( $P3$ ) is involved in both communication segments. There are therefore no incentives for them to attempt such an attack. Similarly, a collusion between the device and the cloud platform ( $P5$ ) provides both parties with all the information that can be gained from this attack.

**Relay attack** This attack is not thwarted by our libraries. The discovery process as well as message issuance and reception is handled by the application code. It is therefore impossible for the libraries to measure the time elapsed between messages, which is usually the indicator of such attacks. The only control that we can provide over this type of attacks is the time out delay associated to the challenge issued for the access request. But setting the cut out time too low would hinder usability.

However, by  $A7$ , the mobile application requires the user to locally authenticate with his phone. So such an attack requires a specific action from the user themselves. The attacker needs more than proximity to their victim.

**Delayed MitM** Access requests include a random challenge that cannot be predicted without breaking assumption  $A2$  (secure random generator). It is therefore not possible to capture a sequence of messages from the mobile application that can later convince the device of the legitimacy of the request. This would require  $\mathcal{A}$  to compute a valid challenge without the corresponding private key, which would break Assumption  $A1$ .

#### 4.5.5.3 Replay attacks

The format and specifics of the token request are out of the scope of our proposal. However, according to  $A9$ , the cloud platform must authenticate a user before issuing a token. Therefore, unless the replayed messages contain an authentication process, the cloud platform should not grant the request. Otherwise, if  $\mathcal{A}$  can successfully replay a token request, the TGL will issue a new token corresponding



to the specifications of the original request. This breaks the confidentiality of the device's state and data, user privileges as well as the integrity of the device's state.

Access requests cannot be replayed. Provided the device has access to a secure source of randomness (Assumption *A2*), challenges prevent the repetition of access request messages. These challenges are issued and verified by the TIL, which code cannot be corrupted (Assumption *A4*).

The device owner (*P2*) and cloud platform operator (*P1*) do not gain any advantage from their involvement in the system.

#### 4.5.5.4 Eavesdropping

According to our definition of eavesdropping, different profiles have access to drastically different information.

Passive eavesdropping cannot reach the secure communication channel over which the private token is transmitted (*A3*). The token is therefore safe while in transit. It is also safe in storage according to *A8*. So under our security assumption, the private token cannot be observed.

The public token however can be observed in transit. In doing so, one can learn what resources are hosted on the device, or the pseudonym of the user. A malicious cloud provider can lift the pseudonymity and observe the usage pattern of users. If the cloud provider is colluding with the IoT device, this information is even more trivial to obtain.

Considering the attacker to be a malicious or curious device owner, the token can leak information when it is presented with the access request. The permissions contained within give indication on the user's interests. If a token contains permissions for more than one device, a malicious device learn information about other devices. The token does not have to contain the subject's identity however, which improves privacy. The lifetime of the token can be modulated to ensure unlinkability of operations.

To protect the token, we can either trust the transport level encryption or encrypt the token at the library level. This would require the TGL and TIL to share yet another key. Library-level encryption of the token does not prevent information leakage from the access request itself. If the token carries a large set of privileges associated to the user, the access request is always specific and identifies which resource the user is exactly interested in.

The use of session keys protects the information exchanged after access has been granted. This prevents the cloud platform in particular to read exchanges between the mobile application and the device. However, if the privileges contained in a token are too narrow, the cloud platform can learn information about the user's interests. To prevent this, tokens can carry a broader range of permissions than those exactly needed to perform their operation. It can mean grouping permissions pertaining to several devices or granting permissions at a higher granularity.

The cloud platform operator is the only one that can link two tokens to the same user. Each token represents a new set of credentials. Lifetime and maximum number of uses should be set in accordance with each use case specific threat model to best preserve the user's privacy and the device's resources.

### 4.5.6 Other security consideration

**Trust model** Our example use case (car rental) placed us within a walled garden ecosystem where a single entity is responsible for the code running on the cloud platform, the mobile, and the device. This is not a requirement but rather a simplification hypothesis. Provided the bootstrap is correctly implemented, the libraries can run on entities operated by different actors.

Privacy becomes more relevant when different actors are involved as other entities may be *honest-but-curious* or simply malicious (Profiles  $P_1$ ,  $P_2$ ,  $P_3$ , and  $P_5$ ). By colluding (or if they are one and the same), the cloud platform operator and the device owner are able to track the user activity and break pseudonymity.

Privileges can be split between tokens (and therefore pseudonyms) to hide user's intentions to devices. For the same reason, a device could be registered with several cloud platforms to prevent a complete mapping of its resources by one actor.

This implies multiple keys or certificate as each cloud platform will perform a separate bootstrap. The device must also track states separately for each cloud platform: for type #1 tokens, both cloud platforms cannot be synchronized on the counter number they associate with each token. If the device only maintains one counter, two tokens can co-exist with the same counter number, both valid as they were issued by a different cloud platform, both associated to a different user with different privileges. To avoid that situation, the device must maintain separated states for each cloud platform.

**Mobile security** The security of mobile phone is an active ongoing topic of research. Our principal concern is the possibility of a malicious application running on the same device. The capacities of this type of attacker are highly dependent on the degree of separation built into the phone. They may share key storage, logs, the app may monitor outgoing connections, see when a request is placed, intercept the token, etc. Under the right circumstances, the malicious application could therefore impersonate the legitimate application and perform a successful privilege elevation attack.

Assumption  $A_6$  places this threat out of our scope. It is nonetheless worth mentioning as  $A_6$  is a strong assumption that is not guaranteed to hold.

**Grouping permissions** Symmetric cryptography tends to multiply keys. To avoid this both on the platform with multiple registered devices and on the device with a multiplicity of tokens, one might be tempted to group devices or permissions.

Grouping devices makes a lot of sense in certain use cases, where a number of devices distributed over an area perform the same tasks and are controlled by the same policies for instance. However, doing so can leak information to curious device owners ( $P_2$ ) or curious users ( $P_3$ ).

Grouping permissions allow for a more compact expression of privileges. A token could carry several sets of permissions involving several devices with different policies. Privacy becomes a concern however as Device  $A$  could learn information about Device  $B$  by reading the token of user  $U$  that has access to both. Information such as the name of the resources it hosts, what client they have in common, what kind of token they accept and therefore what capacity they are likely to have, etc.

But as we have mentioned, ultra-specific permissions also leak information to the cloud platform. A compromise needs to be reached based on the use case and the trust that can be placed in each actor.

## 4.6 Proof of Concept (PoC)

A Proof of Concept based on the proposal described above was developed by Gemalto. A team of five people, including myself, worked on this implementation and the development of the associated *live-coding* video demonstration. The PoC is focused on usability rather than efficiency and is built to showcase ease of integration. Only type #1 tokens were implemented. The implementation of type #2 tokens will be part of a second iteration of this PoC focused on modularity.

The PoC contains an implementation of each of the three libraries as well as sample code illustrating a potential use case. The objective of the PoC implementation was the production of a short 5-minute video aimed at developers highlighting the use and integration of our libraries for each of our three technical actors as well as an example of an access control scenario.

The PoC helped refined our proposal. Some of the specifics presented in the previous sections are therefore not present in the PoC. A crucial example is the absence of the challenge key. Additionally, some of the chosen standards appear ill-fitted in retrospect and will be changed in future iterations. When either of these instances arise, they are discussed.

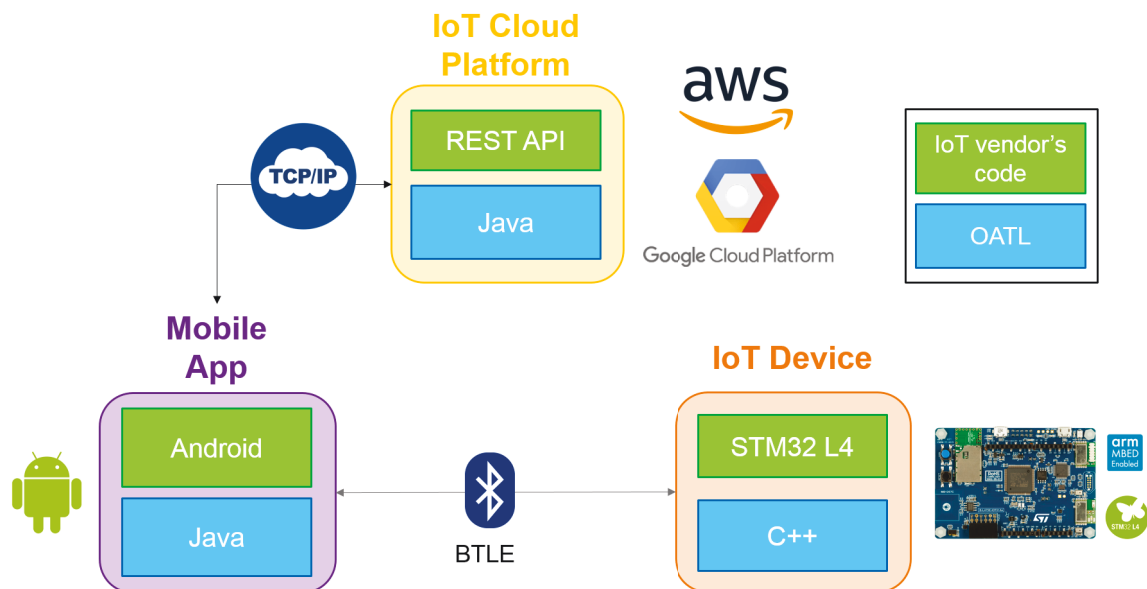


Figure 4.9: OATL implementation architecture

Figure 4.9 presents the communication protocols and implementation languages used in the PoC.

**Use Case Application** We assume the role of a developer working on smart locks for a hotel chain. This use case presents many similarities with the car rental use case, though the latter was more appropriate to emphasize the need for serverless authorization and was therefore preferred to illustrate the project.

We are operating under the same walled garden ecosystem assumption. Clients book their rooms on the hotel's website. They can then retrieve a virtual key to open their room without going to the reception. When the stay is over, the client's access to their room is revoked.

**Cloud platform (TGL)** Two instances of the cloud platform have been deployed: one on AWS, the other on Google cloud. Both use docker containers for portability. Information on devices such as their associated keys and counter numbers are stored in a MySQL database. The cloud platform exposes a REST API that is invoked by the mobile application to generate tokens. No discovery mechanism is proposed. Addresses are hard coded. Similarly, no authorization engine is implemented.

The TGL is implemented in Java. It uses SHA256 [NIST, 2001] for the HMAC. PBKDF2 (Password-Based Key Derivation Function 2 [Kaliski, 2000]) is used for token key derivation with 16384 iterations (more than 10 000, as per the recommendations). A pass-phrase is shared as the PSK between the TGL and the TIL during bootstrap. The token contains a salt as well as the number of iterations required to derive the key.

PBKDF2 is not adapted to that usage. We will prefer NIST SP 800-108 [Chen, 2008] for key derivation in future iterations.

**Mobile application (CSL)** We implemented an android application to use over the CSL. It is written in Java. The GUI of the application is presented in Figure 4.10. It lets the user book a room, which triggers a token request, and lock/unlock the door of said room, which equates to an access request.

Secret keys are stored in the android keystore. No challenge key is derived by the CSL. The token key is used instead.

**IoT Device (TIL)** The STM32L4 Discovery kit <sup>40</sup> is used for the device. TIL is implemented in C++. Communications between the device and the mobile app are encrypted using AES 256 over BTLE.

In the current setting, the key derivation on the device takes over 3 seconds. This represents a noticeable wait for the user. Resource efficiency should be tackled in the next iteration.

## 4.7 Comparison to the state of the art

OATL uses tokens. The specifics of capability-based solutions are discussed in Section 3.2.2. OAuth 2.0 is a popular framework when it comes to tokens. It similarly abstracts the authorization layer and has many available implementations. However, OAuth 2.0 uses bearer tokens, which means that the protocol does not verify the legitimacy of a subject to be using a given token. This works because OAuth 2.0 requires encrypted communications (TLS) for every segment. In the IoT, this poses two problems: First, it binds the framework to one protocol in an environment where communication protocols are varied and ever-changing. Second, it makes security reliant on transport-level security which cannot be assumed in this context. Furthermore, OAuth 2.0 uses a lot of messages exchanged between the different actors which is not compatible with actors with limited bandwidth. Therefore, OAuth 2.0 cannot be applied as-is to IoT use cases.

---

<sup>40</sup><https://www.st.com/en/evaluation-tools/b-1475e-iot01a.html>, Last Checked: July 21st, 2019

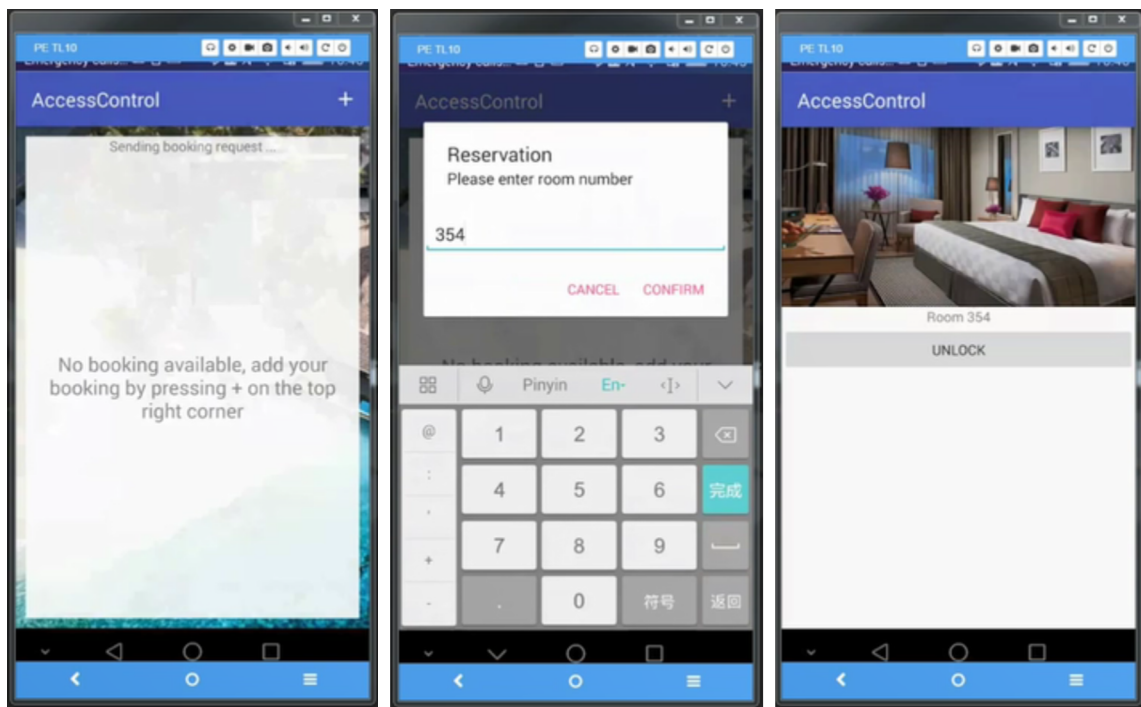


Figure 4.10: Mobile Application GUI

RFC7800 [Jones et al., 2016] addresses the bearer token issue by introducing a Proof of Possession to the OAuth 2.0 standard. In essence, the subject is given (or generates) a key that the authorization server (PDP) either stores or encrypts and embeds in the token. In the first case, the authorization server must be contacted at access time to communicate the key to the IoT device, which forbids serverless authorization. In the second case, the IoT device decrypts part of the token to retrieve the key. This addition does not address the high number of messages required by the OAuth 2.0 framework. Furthermore, in the case of symmetric keys, it is subject to the key escrow issue that is addressed by the introduction of challenge key in OATL type #1 tokens.

Table 4.3 compares our proposal to the solutions presented in the state of the art. Our solution adopts a PDP/PEP hybrid architecture, as described in Section 3.7.1. As such, it shares a few similarities with solutions that use the same architecture: Hybrid solutions are more scalable than fully centralized solutions but less so than solutions opting for multi-PDP or blockchain based architectures (see Sections 3.7.2 and 3.7.3 respectively). Authorization decisions are transmitted using tokens, which enables serverless authorization. The solution provides high context awareness by enabling the evaluation of local conditions at access time. This local evaluation however negatively impacts resource efficiency. For that reason, it is not clear how complex the local evaluation can be. Note that the resource efficiency grade for OATL is based on the PoC implementation. This implementation does not include type #2 tokens. It was also not optimized for resource efficiency. Later iteration will most likely yield a better score in that

area.

When serverless authorization is enabled, they present more resilience as operations can proceed despite the server's unavailability. For the same reason, maintenance operations are easier to conduct on the central server as they do not disrupt operations. Replacing keys and certificates on devices however, is a taxing operation. Permission updates are taken into account when users request a new token.

We share other traits with the solutions from the state of the art that are not due to a common architecture: We operate at a resource-level granularity. We are focused on authorization rather than authentication. Our solution does not mention delegation, nor does it provide auditability. Our proposal includes an implementation in a simulated environment.

Our proposal can be considered as a single-head governance instance when considering the device owner is in control of all aspects of system. If we consider instead that device owner and vendor are two distinct entities, we fall under multi-head governance.

What distinguishes OATL is the use of severless authorization with explicit revocation. Hussein et al. [Hussein et al., 2017a] uses a gateway rather than the object itself as the PDP/PEP. Explicit revocation requires the gateway to be accessible by the server. Bernabe et al. [Bernabe et al., 2016] mentions revocation but does not provide any explicit means of revoking a token once issued. If explicit revocation is only available for type #1 tokens, it is the only proposals that provides it with no connection to the server.

Privacy is also an important feature of our proposal. On one hand, devices don't have access to the identity of users. Each new token represent a new pseudonym. So users cannot be tracked over time. On the other hand, the cloud platform does not have access to the specifics of user transactions as it is not involved in the access request. A user can request a token and never use it. They can also exceed their allotted uses in a few minutes and only request another token a month later. Privacy is set to high instead of very high in Table 4.3 because we have no control over the vendor's code.

Finally, our proposal is focused on usability. It is therefore natural that it scores higher than other solutions in this category.

## 4.8 Conclusion

**Summary** We have presented our work on Offline Authorization Token Libraries. These libraries are meant to help developers tackle security requirements while providing serverless access to users.

Our libraries offer some guarantees of security while other aspects are left to the application side. The guarantees offered by our libraries are:

- Authentication (in the sense that the token can only be used by its intended recipient),
- Authorization (enforcement),
- Anti-replay protection on authorization,
- Session-Key generation for secure communication,
- Secure storage of secret keys on the mobile,

Criteria	OATL	[Hemdi and Deters, 2016]	[Bernabe et al., 2016]	[Hussein et al., 2017a]	[Hernández-Ramos et al., 2013]	[Cert, 2015]	[Seitz et al., 2013], [Cherkaoui et al., 2014]
Model	-	ABAC	ABAC, TBAC	ABAC	-	-	-
Crypto	SHA256, AES, ECDSA	-	ECC	-	ECDSA	asymmetric	AES
Standards	JSON	CoAP	CoAP, XACML, JSON	CoAP, JSON	CoAP, JSON	-	CoAP, DTLS, XACML, SAML, JWE
Use Case	Car rental	-	-	-	-	-	-
AuthN or AuthZ	authZ	authZ	authZ	authZ	authZ	authZ	authZ
Policy storage	-	on device	on PDP	on PDP	-	on device	on PDP
Bootstrap	yes	no	no	no	no	no	yes
Resource Efficiency	very poor (token 1)	very poor	very poor	poor	very poor	very poor	very poor
Resilience	fair	fair	high	high	fair	fair	poor
Serverless	yes	no	yes	yes	yes	no	no
Scalability	fair	poor	high	very high	fair	poor	fair
Maintainability	fair	poor	fair	high	high	poor	fair
Permission Updates	LoT	imm	LoT	LoT	LoT	NTR	NTR (single use)
Usability	very high	poor	fair	high	fair	poor	high
Granularity	RL	-	RL	RL	RL	DL	RL
Context-awareness	high	fair	very high	high	high	very poor	high
Revocation	impl and expl	-	impl and expl	impl and expl	impl	-	impl
Delegation	-	-	-	-	-	-	-
Auditable	-	yes	-	-	-	-	-
Privacy	high	-	fair	poor	very poor	high	poor
Governance	single-head	single-head	single-head	multi-head	-	single-head	multi-head
Maturity Level	SE	SE	SE	SE	SE	Th	SE

“-”: Not Mentioned or Non Applicable - DL: Device Level - RL: Resource Level - expl: explicit - impl: implicit - authN: authentication - authZ: authorization - LoT: Lifetime of Token - SE: simulated environment - NTR: Next Token Request - Th: Theoretical

Table 4.3: Comparison of OATL with the state of the art

- Protection against Delayed Man in the Middle.

OATL does not take care of:

- User authentication,
- Authorization decisions,
- Secure storage of keys for the server and device,
- Mutual authentication,
- Man in the Middle Attacks (relay attacks).

We advocate using a secure enclave on the device and an HSM on the server. The choice of authorization engine is left entirely to the IoT vendor. Additional local conditions can be included in the tokens. These conditions should be expressed in a JSON format. Their interpretation is then left to the application.

**Perspectives** The perspectives for the OATL project boil down to two things: working on a better implementation and going beyond what was proposed in this Chapter.

The PoC presented in Section 4.6 is a first iteration. In future iterations, we would like to:

- Re-implement type #1 tokens with the revised specification:
  - New derivation algorithm,
  - Challenge key,
  - Clearer derivation of session keys.
- Implement type #2 tokens,
- Focus on resource efficiency,
- Test local conditions,
- Test our implementation in a more realistic environment.

To improve on our proposal, we would like to:

- Propose a binary format for type #1 tokens instead of JSON. The JSON format is verbose. A binary format would be more restrictive but would result in a lower overhead.
- Add more token format to deliver on the modularity aspect of the project.
- Offer the user the option to not use the mobile application and instead use their phone as a relay between the device and the cloud platform. It is harder and harder to convince clients to install apps. An alternative route would enhance usability. This would however disable serverless authorization. But it would be the user's choice to do so.



- Propose explicit revocation for type #2 tokens. An idea would consist in embedding revocations inside authorization tokens and use clients to deliver them to the device. A compact representation of revoked tokens is however required.
- Work on delegation aspects. Users should be able to transfer permissions without going through a central entity.
- Discuss auditability. Devices are constrained memory-wise and cannot be supposed to connect to the cloud platform after the initial bootstrap phase. Without a gateway, it is interesting to explore how and where to store access logs for the device.



## Chapter 5

# MAAC-B: Multi-endorsed Attributes for Access Control using the Blockchain

### Contents

---

<b>5.1</b>	<b>Introduction</b> . . . . .	<b>142</b>
<b>5.2</b>	<b>Architecture, Assumptions and Overview</b> . . . . .	<b>144</b>
5.2.1	Actors . . . . .	144
5.2.2	Security Assumptions . . . . .	145
5.2.3	Access Control Scenario . . . . .	146
<b>5.3</b>	<b>Trust Anchor Service</b> . . . . .	<b>148</b>
<b>5.4</b>	<b>Policy Service</b> . . . . .	<b>149</b>
5.4.1	Policy contracts (PolC) . . . . .	149
5.4.2	Dispatch contracts (DisC) . . . . .	152
5.4.3	Policy Retrieval . . . . .	152
<b>5.5</b>	<b>Attribute Service</b> . . . . .	<b>153</b>
5.5.1	Requestor's Identity . . . . .	153
5.5.2	Attribute Contracts (AttC) . . . . .	154
5.5.3	Attribute Publication . . . . .	154
5.5.4	Attribute Revocation . . . . .	155
<b>5.6</b>	<b>Access Control</b> . . . . .	<b>155</b>
5.6.1	Requestor Authentication . . . . .	155
5.6.2	Endorsement Retrieval . . . . .	156
5.6.3	Trust Level Computation . . . . .	156
5.6.4	Access Decision . . . . .	157
<b>5.7</b>	<b>Security Analysis</b> . . . . .	<b>158</b>
5.7.1	Identity Theft . . . . .	158
5.7.2	False attributes endorsements . . . . .	158

5.7.3	Privilege Elevation . . . . .	159
5.7.4	Replay attacks over endorsement transactions . . . . .	159
5.7.5	Replay attacks over policy transactions . . . . .	160
5.7.6	Privilege suppression . . . . .	160
5.7.7	Other issues and mitigation . . . . .	160
<b>5.8</b>	<b>Comparison to the state of the art . . . . .</b>	<b>161</b>
<b>5.9</b>	<b>Conclusion . . . . .</b>	<b>162</b>

---

## 5.1 Introduction

The previous chapter presented a PEP/PDP hybrid solution with a centralized server. We go further towards distribution by doing away with this central entity and replacing it by a blockchain. The benefits of doing so have been elaborated on in Section 3.7.3.

MAAC-B (Multi-endorsed Attribute for Access Control using the Blockchain) defines an attribute management system based on blockchain transactions. In MAAC-B, attributes are stored in smart contracts that are deployed and managed by the users themselves. These attributes can be issued by anyone with a single blockchain transaction. A trust level is then computed for each attribute based on the entities that have endorsed it.

MAAC-B also provides a policy management system where administrators can define generic policies and assign them to resources held by IoT devices under their control. The blockchain is used to issue, update, and disseminate these policies.

The objective of this proposal is to provide a distributed attribute-based access control solution that enables interoperability, supports the dynamicity of the IoT (user or device addition, flexible policies, task re-assignment, etc), and gives users control over their identity within the system.

A shorter version of this solution was presented in IWCMC 2019 [Dramé-Maigné et al., 2019]. The original paper limited its proposal to attribute management. The present chapter expands on this work by introducing policy management.

**Use Case** For illustration, we consider the case of a hotel company with locations all around the world. Each hotel hosts a vast array of services including restaurants, pool, spa, gym, etc. These services are available not only to hotel clients but also to the local crowd. In particular, gyms are operated by independent companies that have locations both within and outside the hotels. Both the hotel and the gym companies want to control access to their own infrastructures (i.e., hotel rooms, lockers, gym equipment) independently through smart locks. Having different sets of requirements, different clients, and different interests, neither party is willing to delegate access control to the other.

In this use case, the access control solution needs to be *interoperable* in order to serve both the company operating the hotels, and the ones operating the gyms. As the length of stay varies greatly and new clients can come in at any time, the system should accommodate the *dynamic addition of users*. As a client-centric business, the hotel would benefit from a *user-centric* system. In the spirit of user-centricity, users’ personal information should be treated with care. The system therefore needs to be

*privacy-focused*. It should not sacrifice *ease of management* for the administrators as it is important for them to be able to get a view of room occupancy and allotted privileges. In the event of a network outage, clients should still be able to access their rooms and move around the hotel. *Local access*, where devices only rely on connection to a local manager should therefore be enabled. The system is also expected to *scale* to manage clients coming from the different hotel and gym locations.

**Selection of ABAC and blockchain technology in our solution** The state of art does not fulfill our smart hotel use case requirements (see Section 5.8). We propose an attribute-based blockchain-based approach to access control that provides interoperability, local access, dynamic user addition, contextual authorization, user privacy, and ease of management. The Attribute-Based Access Control (ABAC) model [Hu et al., 2013] enables high expressiveness, in particular in devising context-aware policies. It is compatible with dynamicity in both users and resources, enabling interoperability between federated entities, while allowing for a more compact expression of access policies.

We selected the blockchain technology for:

- keeping track of users' attributes through *per user* smart contracts where attributes can be endorsed by multiple Attribute Issuing Entities (AIE),
- managing the trust level assigned to endorsing AIE by administrators, thus enabling the (smart lock) devices to assess the trust level of an attribute,
- issuing and managing access control policies based on subject's attributes.

Note that reading attributes and policies from the blockchain is a fast operation that is compatible with real-time applications. Endorsing a subject's attribute is a matter of one blockchain transaction. Same goes for modifying the trust level of an AIE, or publishing a policy. As such, the blockchain-based approach supports dynamic addition and removal of both requesting subjects (next referred to as requestors) and AIE.

**Contribution** The contributions of this chapter can be summarized as follows:

- The definition of user-controlled identities that can then be used to separate attributes in accordance with the user's wishes.
- An attribute endorsement system that is entity-independent and remains operational through changes in organizations or partnerships.
- A trusted entities management system enabling administrative changes throughout the life of IoT devices.
- An attribute evaluation scheme based on endorsements and reputation.
- A distributed policy management system where policies can be re-used for multiple resources.
- An overall user-centric access control system enabling interoperability and flexibility.

**Organization** The remainder of this chapter is organized as follows: Section 5.2 discusses the architecture of our proposal. Section 5.3 describes the management of administrators and trust. Section 5.4 details the issuance and management of access control policies. Section 5.5 focuses on attribute endorsement, and Section 5.6 on access control and attribute evaluation. Section 5.7 analyzes the security of the

proposal. Section 5.8 compares our proposal to the state of the art presented in Chapter 3. Section 5.9 concludes this chapter.

## 5.2 Architecture, Assumptions and Overview

This section presents the actors of our solution, our security assumptions, and the high level process of policy issuance, attribute endorsement and evaluation, as well as access control.

### 5.2.1 Actors

Our system includes: IoT devices, administrators, blockchain nodes, gateways, Attribute Issuing Entities (AIE), and requestors. Additionally, we consider an independent reputation system. Going back to the smart hotel use case, clients (requestors) wish to activate smart locks (IoT devices) to access their rooms, the gym, or any door in the building. Gateways are connected to the blockchain and routinely update smart locks on policy or administrative changes. When they receive a request, smart locks query the gateway to retrieve attributes issued by multiple AIE. Smart locks compute each attribute's trust level and evaluate the request based on the requirements defined in the policy, thus deciding whether to open or close the door.

**IoT Devices** IoT devices, behaving either as sensors or actuators, decide whether a requestor is permitted to access some resources (e.g. sensed data), or to perform an action.

The device is configured with the addresses of two blockchain smart contracts: one is used to manage its trust, the other associates a device's resources with access control policies. When the device is associated to a single long-term gateway, this configuration can be hosted there.

The access control decision requires the retrieval of the appropriate policy, the extraction of attribute's endorsements, and their verification in order to determine their associated trust level. The device acts both as a PEP and a PDP by evaluating attributes.

**Blockchain** The blockchain supports three services:

- The *Trust Anchor Service* is made up of transactions published by the administrators to promote new administrators, add entities to the list of trusted entities, or update the reputation score of an entity. These transactions are discussed in Section 5.3.
- The *Policy Service* is comprised of all transactions that relate to policy publication or update. Policies define the rules that govern access in the system and involve requirements for requestor's attributes. These transactions are introduced in Section 5.4.
- The *Attribute Service* is comprised of all transactions published by the AIEs to endorse or revoke attributes. These mechanisms are detailed in Sections 5.5.

The blockchain serves as a vehicle that stores and transports access control policies. It also acts as a PIP with regards to attribute retrieval.

**Reputation System** The reputation system operates independently. It can be hosted on the blockchain or outside of it. We will consider it as a black box. It provides reputation scores for all types of entities that are then normalized between 0 and 1.

**Gateway** Due to considerable resources being consumed by the blockchain, connectivity to IoT devices still remains a big challenge, and an ongoing research topic (see Section 2.10). Until an optimized solution is found, we introduce a gateway to act as a proxy between the blockchain and IoT devices. Each device is therefore registered with a gateway that takes part in the blockchain network and filters new blocks, extracting data relevant to each device.

Through gateways, devices retrieve administrative or policy updates either via periodical updates or at their own request. Update periodicity is defined by administrators.

Additionally, the gateway understands a large range of communication protocols. IoT devices are therefore not required to be IP-enabled. They communicate with their gateway using short range protocols such as Bluetooth Low Energy (BTLE). As such, in the case of a network outage, communication with the gateway are still operational. This enables local access.

A device can be associated to a single gateway. Alternatively a device may be registered with several gateways or dynamically register with new gateways over the course of its life. The latter is particularly indicated for devices with location that varies over time. In this case, a dynamic registration process must be devised that is out of the scope of this proposal.

**Administrators** Administrators perform administrative operations through the *Trust Anchor Service* and *Policy Service*. They must therefore have a blockchain address and be able to publish transactions. Each device is associated to at least one administrator.

**Attribute Issuing Entities (AIE)** AIEs endorse attributes of requestors according to their own policy (which is out of scope of our contribution). They are required to have blockchain capabilities.

**Requestors** The entity requesting access to a device's resources and required to be provided with some attributes might be either a device itself, or a more powerful entity such as a desktop or a mobile phone. Requestors may act independently or on behalf of a user.

### 5.2.2 Security Assumptions

We operate under the following assumptions:

- A1 *Secured blockchain keys*: Blockchain keys cannot be stolen, lost or otherwise compromised.
- A2 *Solid cryptographic primitives*: Signatures, hashes and other cryptographic primitives cannot be broken.
- A3 *Blockchain consistency over time and nodes* [Pass et al., 2017]: All nodes in the network agree on blockchain history, the last few blocks excluded. Accepted transactions cannot be modified.

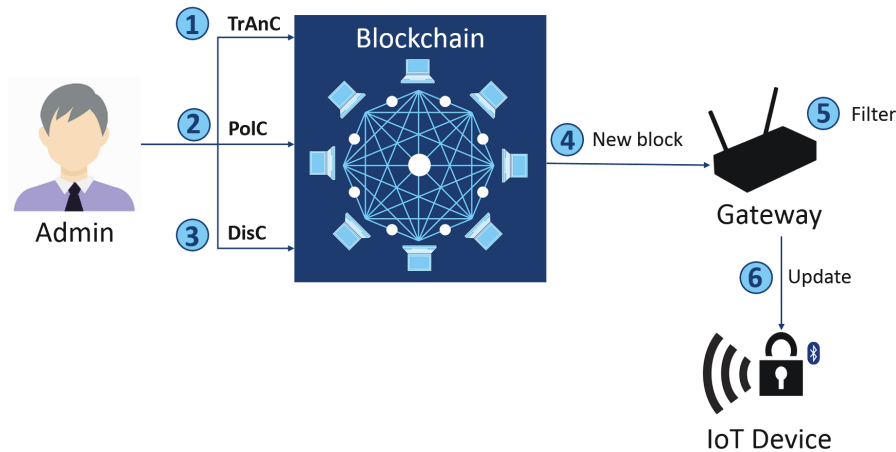


Figure 5.1: Access policy issuance and update

A4 *Blockchain growth* [Pass et al., 2017]: Valid transactions will integrate the blockchain eventually.

A5 *Trusted and Reliable Gateway*: The gateway is assumed to be trust-worthy and accessible.

A6 *Trusted Entities (TE)*: Administrators and AIEs designated as TEs are trusted.

A7 *Robust Random Generator*: Devices are provided with robust random generators.

A8 *Secure Reputation System*: The reputation system is considered secure.

### 5.2.3 Access Control Scenario

Our access control scenario is composed of two processes that happen in parallel: the management and retrieval of access policy and attributes. For both, we exclude the bootstrap phase that is considered out of scope. It entails: the creation of blockchain accounts, the pairing of each IoT device with a gateway, the configuration of IoT devices (i.e. keys, contract addresses, ...).

The process by which AIEs decide to issue attributes for a requestor is out of scope as it is AIE-dependent. In the smart hotel use case for instance, an attribute is issued when a client signs up for a gym membership. The discovery process between the requestor and the device is also out of scope.

We present the high level procedures of policy and attribute management below.

**Access policy management** Figure 5.1 presents the high level steps required to issue and revoke policies. First, administrators publish a Trust Anchor Contract (TrAnC) to establish the trust relationships for their devices (step 1). After that initialization, administrators publish both Policy Contracts (PoIC) and Dispatch Contracts (DisC) (step 2& 3). The former manages policies, the latter links policies to resources. Resources are hosted on IoT devices. Several devices can host the same types of resources. For



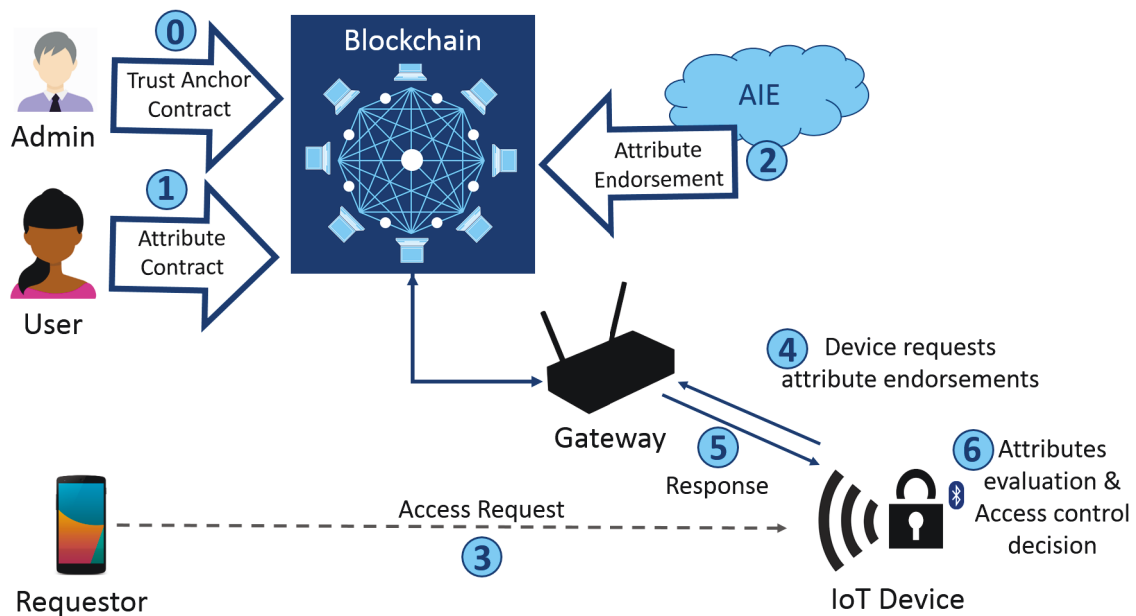


Figure 5.2: Attribute management and evaluation

instance all smart thermometers will host a *temperature* resource. This lets administrators group similar devices under the same policy.

When a new block is created (step 4), the gateway scans it for policy updates relevant to the devices that are under its care (step 5). If changes occurred, the gateway notifies the corresponding device(s) (step 6). Updates are sent according to the availability of the device and the update frequency defined by administrators.

**Attribute management** Figure 5.2 presents a high level illustration of how the different actors work together to manage and verify attributes. First, administrators establish the trust relationships for their devices (step 0). In parallel, the user or somebody acting on its behalf deploys an attribute contract (step 1). Using this smart contract, one or several AIE endorse the appropriate attributes (step 2). Endorsements can be revoked at any time by the AIE that issued them. They are the only one allowed to revoke endorsements.

When a client requests a smart lock's opening (step 3), the device contacts its gateway (step 4 & 5) to retrieve attribute endorsements. Finally (step 6), the smart lock evaluates them against the applicable policy and makes its decision.

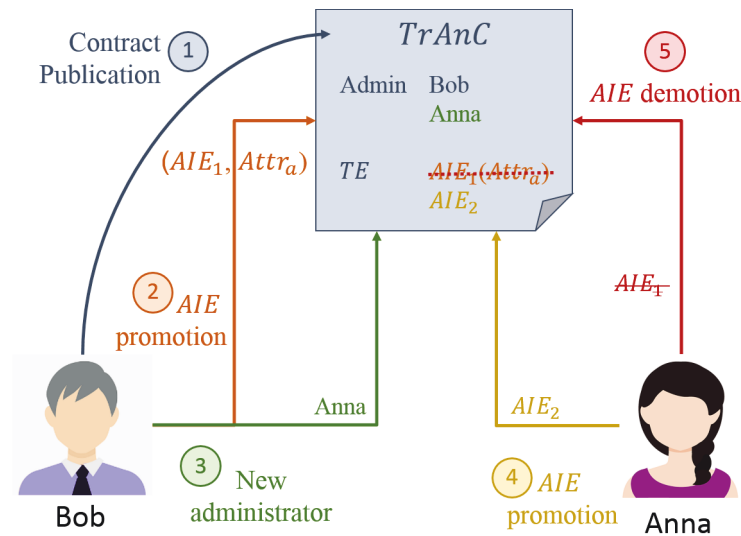


Figure 5.3: Trust Anchor Contract

### 5.3 Trust Anchor Service

The blockchain opens a communication channel between administrators and devices through Trust Anchor Contracts (TrAnC). Through these contracts security updates are pushed to devices, including the list of permitted administrators, the list of trusted AIEs, and AIEs' reputation scores. These reputation scores are an addition to the reputation system. Administrators have no control over the latter. This lets them overwrite decisions they may disagree with, either increasing or lowering the score of an entity. The invocation of state-modifying functions is reserved to administrators. The read operations are open to all.

TrAnC must implement the following functions, summarized in Table 5.1:

- *isAdmin*: to verify that a given blockchain address is defined as an administrator in this contract.
- *newAdmin*: to add a new administrator in the contract.
- *removeAdmin*: to remove an administrator. Combine with *newAdmin* to change the blockchain address of an administrators after devices have been deployed.
- *newTrustedEntity*: to add a new AIE to the list of TE. The second argument (scope) is optional and restricts the trusted status of an AIE to these attributes only. It is formatted as a string containing attributes separated by colons,
- *removeTrustedEntity*: to revoke an AIE TE status.
- *getTrustedEntitiesList*: to get the list of TEs with TE's scope when applicable.

Function	Type	Arguments	
isAdmin	call	<i>address</i>	blockchain address of the entity
newAdmin	tx	<i>address</i>	blockchain address of the new administrator
removeAdmin	tx	<i>address</i>	blockchain address of the revoked administrator
newTrustedEntity	tx	<i>AIEPubKey</i>	public key of the AIE
		<i>scope</i>	<i>Opt.</i> attributes for which the AIE is to be considered as a TE
removeTrustedEntity	tx	<i>AIEPubKey</i>	public key of the AIE
getTrustedEntitiesList	call	-	-
updateReputationScore	tx	<i>AIEPubKey</i>	public key of the AIE
		<i>score</i>	new reputation score for this AIE.
getReputationScores	call	-	-
deleteContract	tx	-	-

*tx (short for transaction): state-modifying functions; call: read-only functions*

Table 5.1: Functions of an Trust Anchor smart Contract (TrAnC)

- *updateReputationScore*: to update reputation of AIEs (after an AIE has been compromised for instance). A negative score means that attribute endorsements from this AIE should be disregarded
- *getReputationScore*: to get the reputation score of an AIE.
- *deleteContract*: to destroy the TrAnC and all of the included information.

Smart contracts should always include a function to remove the contract from the blockchain. This prevents blockchain bloating. A contract can never be deleted otherwise.

Figure 5.3 illustrates the use of a TrAnC. The user that deploys the contract (Bob in step 1) is considered as an administrator by default. As such, Bob can promote AIE to a Trusted Entity status (step 2). When a scope is provided (here,  $Attr_a$ ), trust in the AIE is limited to that scope. TrAnC can also be used to designate new administrators (step 3). Once a user has been declared as an administrator, all functions are accessible to them, including AIE promotion (step 4), and the demotion of AIE promoted by other administrators (step 5).

## 5.4 Policy Service

The *Policy Service* is made up of two smart contracts. Policy contracts (PolC) house policies associating attributes and trust level. Dispatch contracts (DisC) assign resources to policy contracts. Together they define which attributes a requestor must possess (and the required trust level) to access a resource.

Each PolC and DisC is associated with a TrAnC which *isAdmin* function is called each time a state-modifying function is invoked.

### 5.4.1 Policy contracts (PolC)

PolC are a collection of policies, each of which is a list of requestor's attribute and a corresponding trust level. These policies are defined in a resource-agnostic fashion. This enables reusability which lowers

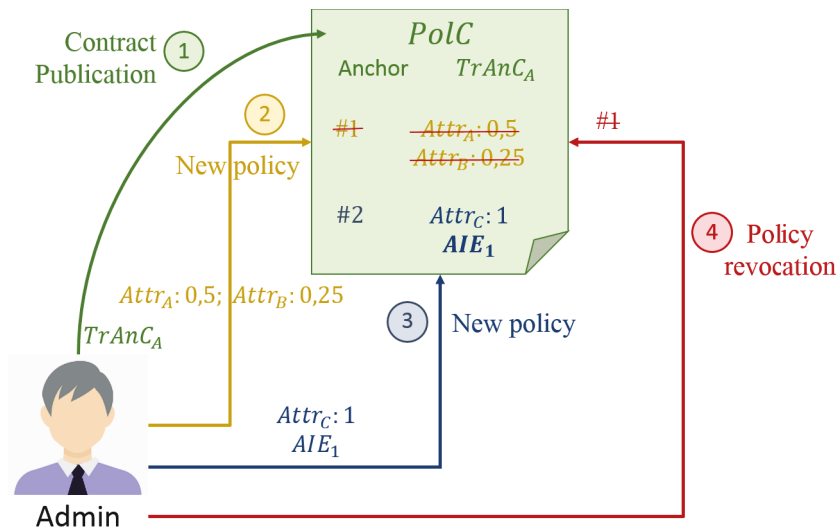


Figure 5.4: Policy Contract (PolC)

the complexity of the system.

The following functions must be implemented in PolCs. They are summarized in Table 5.2.

- *addPolicy*: to add a policy. Input must be formatted as such:

$$Attr_A : TL; Attr_B : TL; \dots Attr_U : TL$$

where  $Attr_A$  is a requestor attribute and  $TL$  is the associated trust level. Optionally, policies can designate a Trusted Entity. On creation, a policy is allotted the next available Policy Number.

- *updatePolicy*: to update a policy already associated with a Policy Number. It takes the same input format as *addPolicy*, including an optional AIE.
- *removePolicy*: to remove a policy given its Policy Number.
- *getPolicy*: to retrieve a policy based on its Policy Number. The returned policy is formatted as such  $(policyNumber, Attr_A : TL; Attr_B : TL; \dots Attr_U; TL, TE)$
- *getAllPolicies*: to retrieve all the policies stored within the PolC. The returned policies are formatted as such  $policy_1, policy_2, \dots, policy_i$  where  $policy_i$  is formatted as showcased in *getPolicy*.
- *deleteContract*: to destroy the PolC and all of the included information

Figure 5.4 provides an example of the use of a PolC. First, an administrator deploys the contract (step 1). Contrary to TrAnC, the user that issues the blockchain transaction creating the contract does

Function	Type	Arguments	
addPolicy	tx	<i>policy</i>	string of the form $att_A : TL; att_B : TL$ ; representing a policy
		<i>AIEPubKey</i>	Opt. public key of an AIE that will be a TE for this policy
updatePolicy	tx	<i>policyNbr</i>	the number of the target policy in the policy table
		<i>policy</i>	new policy (see addPolicy format)
		<i>AIEPubKey</i>	Opt. new TE
removePolicy	tx	<i>policyNbr</i>	the number of the target policy in the policy table
getPolicy	call	<i>policyNbr</i>	the number of the target policy in the policy table
getAllPolicies	call		
deleteContract	tx	-	-

*tx* (short for transaction): state-modifying functions; *call*: read-only functions; *TL*: Trust Level

Table 5.2: Functions of a Policy smart Contract (PolC)

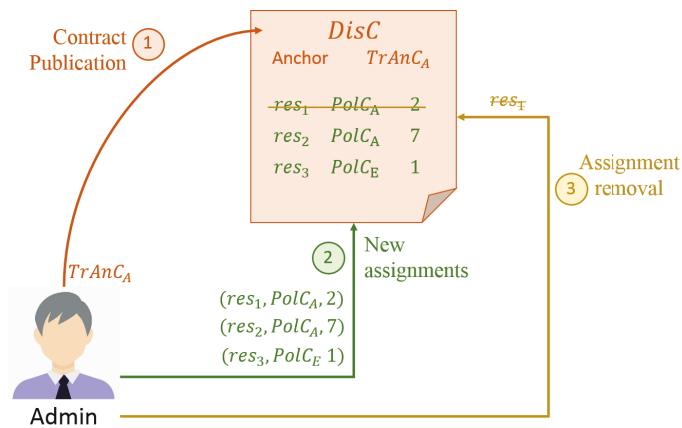


Figure 5.5: Dispatch Contract

not gain administrative privileges over it. Instead, the blockchain address of a TrAnC must be provided (here  $TrAnC_A$ ) to manage the rights to issue and revoke policies.

An administrator (a user designated as such in  $TrAnC_A$ ) then issues a new policy (step 2). This policy involves two attributes ( $Attr_A, Attr_B$ ) with different associated trust levels (0.5 and 0.25 respectively). The contract returns a policy number (here 1).

Another policy is published (step 3) with a designated trusted entity ( $AIE_1$ ). This means that for the purpose of this policy, attributes endorsed by  $AIE_1$  will be given a trust level of 1. The policy only involves one attribute ( $Attr_C$ ) with a required trust level of 1.

Finally, the administrator revokes policy #1.

Function	Type	Arguments	
addAssignment	tx	<i>res</i>	resource name
		<i>polAddr</i>	address of the PolC
		<i>polNumber</i>	number of the target policy
updateAssignment	tx	<i>res</i>	resource name
		<i>newPolAddr</i>	address of the PolC
		<i>newPolNumber</i>	number of the target policy
removeAssignment	tx	<i>res</i>	resource name
getAssignment	call	<i>res</i>	resource name
deleteContract	tx	-	-
<i>tx (short for transaction): state-modifying functions; call: read-only functions</i>			

Table 5.3: Functions of a Dispatch smart Contract (DisC)

### 5.4.2 Dispatch contracts (DisC)

DisC associate resources and policies. Each device is associated with one or several DisC. The following functions must be implemented in DisC. They are summarized in Table 5.3.

- *addAssignment*: to associate a resource with a policy designated by a PolC address and a policy number.
- *updateAssignment*: to update an already existing association between a resource and a policy. If the given resource has not been assigned to a policy yet, it should fail.
- *removeAssignment*: to remove the association between a resource and a policy
- *getAssignment*: to retrieve the policy associated with a given resource. The output will be of the form (*res*, *PolC*, *polNbr*).
- *deleteContract*: to destroy the DisC and all included information.

Figure 5.5 provides an example of a Dispatch Contract. As with PolC, DisC take the address of a TrAnC at the time of deployment (step 1). The contract is then used to store assignments of policy to resources (step 2). A resource is assigned to a single policy, identified by the address of the Policy Contract and its Policy Number. The same policy can be assigned to several resources. When validating an assignment, the DisC should retrieve the corresponding policy to check that it exists.

To remove an assignment (step 3), the administrator provides only the name of the resource. To minimize transactions, policies in PolC should be updated rather than revoked and re-issued. This dispenses administrators from re-issuing assignments whenever a policy is updated.

### 5.4.3 Policy Retrieval

Devices receive policy updates from their gateway(s). The gateway, being a blockchain node, updates its own ledger each time a new block is mined. The gateway gathers transactions pertaining to TrAnC

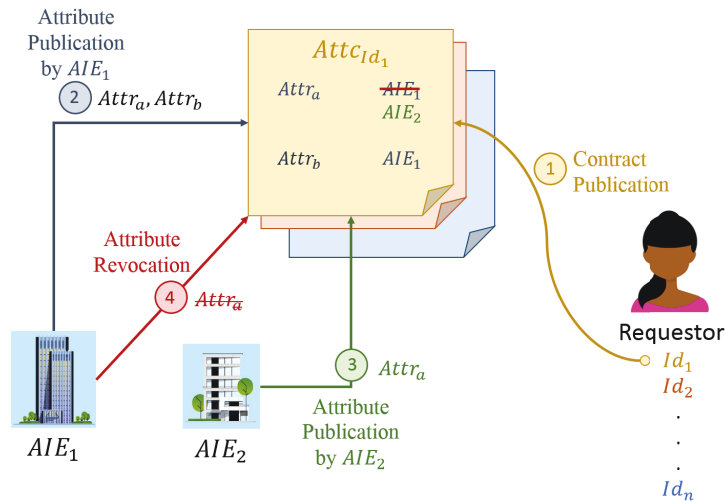


Figure 5.6: AttC: Attribute endorsement and revocation

or DisC associated to any of its devices. It also follows modifications to PoIC. As there are no links between PoIC and the DisC that assign them to resources, policy revocations and modifications are not automatically pushed to DisC. When a resource is assigned a new policy, the gateway queries the corresponding PoIC to retrieve it.

The gateway gathers these updates and either delivers them to the concerned device or stores it to be provided on demand.

## 5.5 Attribute Service

Attribute publication steps are described in Figure 5.6. Each step is defined below.

### 5.5.1 Requestor's Identity

Requestors can have one or several identities that take the form of a public/private key pair. The identifier in itself is a hash of the public key. It is neither provider-dependent, nor blockchain-dependent and can be created autonomously and at will. Attributes are linked to the subject through one of its identities.

A subject can therefore split its attributes amongst different identities. Attributes could be separated by use: one id for work, one id for government-issued attributes, one id for vacation purposes, etc. They could be separated by issuer, or for any other reason. The user is in control of what identity they provide to each AIE while looking for endorsements. This enhances privacy by letting users switch pseudonyms.

### 5.5.2 Attribute Contracts (AttC)

AttC are used to link attributes to requestors. One (or more) AttC must be deployed for each requestor. The identity of its associated requestor is stored *in* an AttC and used to validate endorsements. It *cannot* be modified. Deployment of the AttC is the first step of Figure 5.6. For each attribute, the smart contract stores a list of AIEs that have issued endorsements, and their signatures.

Function	Type	Arguments	
endorseAttribute	tx	<i>attr</i>	name of the attribute to endorse
		<i>AIEPubKey</i>	public key of the issuing AIE
		<i>blockNumber</i>	block number at the time the transaction was issued
		<i>sign</i>	signature of the hash of <i>id  attr  blockNumber</i> , where <i>id</i> is the identity of the requestor
removeEndorsement	tx	<i>attr</i>	name of the attribute to revoke
		<i>AIEPubKey</i>	public key of the issuing AIE
		<i>sign</i>	signature of the hash of <i>id  attr  blockNumber  revoke</i> , where <i>id</i> is the identity of the requestor and <i>blockNumber</i> was the argument used at the time of issuance
getEndorsements	call	<i>attr</i>	name of the attribute to confirm
deleteContract	tx	-	-
<i>tx (short for transaction): state-modifying functions; call: read-only functions</i>			

Table 5.4: Functions of an Attribute smart Contract (AttC)

There are no restriction on who can invoke an AttC's functions, *deleteContract* excluded. The latter can only be invoked by the contract's owner, i.e. the identifier embedded in the contract. Attributes can be endorsed by any entity. The following functions must be implemented in AttCs. They are summarized in Table 5.4.

- *endorseAttribute*: for AIE to endorse attributes.
- *removeEndorsement*: to revoke an endorsement.
- *getEndorsements*: to retrieve an attribute endorsement set which include a list of endorsers and the material necessary to verify their signatures.
- *deleteContract*: to destroy the AttC and all the information it contains.

### 5.5.3 Attribute Publication

Once an AttC is deployed, AIE can start issuing attributes for the associated identity. This is represented as steps 2 and 3 of Figure 5.6.

In order to do so, an AIE must first retrieve the AttC's address. A requestor most likely has to interact with an AIE one way or another before the AIE agrees to endorse an attribute. During this exchange, the AIE retrieves the AttC's address and the requestor's identity. This exchange is out of scope and each AIE is free to implement its own process for authentication and attribute verification.



The AIE then issues a blockchain transaction invoking the *endorseAttribute* function of the AttC. When the function is called, the AttC first reconstructs the signed message using the provided argument and the stored requestor's identity, and checks the provided signature against the provided public key. This operation applies to non-endorsed attributes (step 2 of Figure 5.6) and attributes with pre-existing endorsements (step 3 of Figure 5.6). For each attribute in the AttC, an endorsement record should include the following elements:

*attr* : (*AIEPubKey*, *blockNumber*, *sign*)

#### 5.5.4 Attribute Revocation

An AIE is always free to revoke its attribute endorsements. To do this, it must invoke the *removeEndorsement* function of the corresponding AttC. This is depicted as step 4 of Figure 5.6.

After verifying the signature and checking it against the stored information, the AttC removes the AIE's endorsement. If the attribute is then no longer endorsed by any other AIE, the attribute is removed from the AttC. If other AIEs have issued endorsement for this attribute, the attribute remains.

Note that AIE are responsible for the attributes they issue. As such, they are assumed to keep track of issued attributes along with the associated identity and address of the smart contract that was used.

## 5.6 Access Control

We consider that before the access control process begins, the latest policy updates have been delivered to the device. This means that the Gateway sorted through new blocks, identified changes in either PolC or DisC related to the device, and sent the corresponding modifications. The security policies are therefore hosted on the device itself and do not need to be retrieved at access time.

Before granting access to resources, the device needs to evaluate the requestor's attributes against the corresponding security policy. Attribute evaluation can be separated into three steps. An access request must be placed and associated attributes identified. The corresponding endorsements need to be retrieved from the blockchain. Finally a trust level must be assigned to each attribute. The attribute retrieval process is illustrated in Figure 5.7.

### 5.6.1 Requestor Authentication

An access request (step 1 (a) of Figure 5.7) must include the requestor's identity, the address of the associated AttC, and the public key linked to the identity. A challenge message is then sent to authenticate the requestor. If the signature of the challenge matches the provided public key, the IoT device retrieves the policy corresponding to the access request.

Alternatively, a client may choose to use a combination of identities to prove that it possesses all the required attributes (Step 1 (b) of Figure 5.7). In that case, and if the device supports it, the access request should not carry any identifying information. The device first retrieves the policy corresponding to the request and extracts the required attributes. They are then sent to the client along with a challenge message. For each attribute, the client must provide an identity, the corresponding AttC address, public key, and signed challenge.

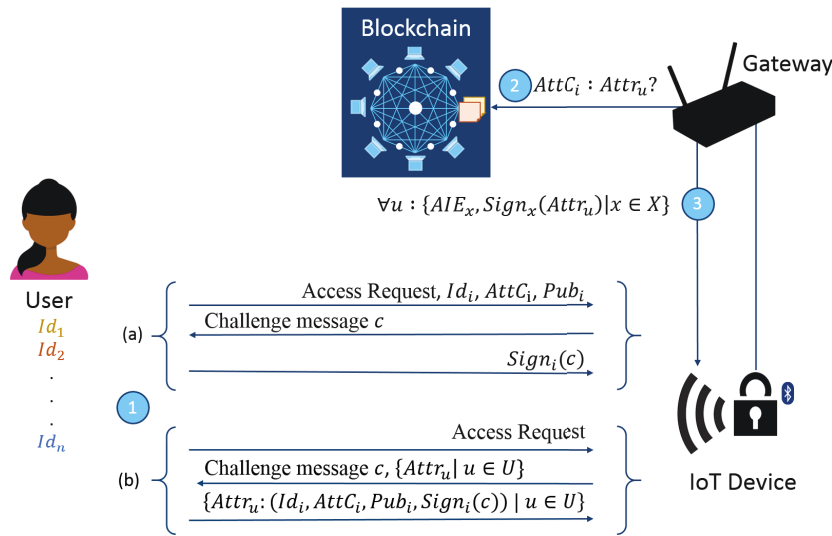


Figure 5.7: Attribute Retrieval

For this alternate authentication, the device needs to interrogate several AttC instead of one in order to retrieve the attributes’ endorsements. This however amounts to the same number of AttC requests as one needs to be placed for each attribute. However a greater number of signatures must be verified to authenticate the client. This option should therefore be limited to IoT devices that can handle the increase in computational requirements.

Though this feature adds flexibility to the system, there are some privacy concerns: Any requester can obtain the list of attributes necessary to gain access to a resource. In settings where this is too much to share, this option should be disabled.

### 5.6.2 Endorsement Retrieval

Attributes are retrieved based on the applicable access control policy. For each of these attributes, the IoT Device queries the AttC via the gateway, using the *getEndorsements* function (step 2). In response, the device receives a set of endorsements for each attribute as shown in step 3 of Figure 5.7.

Endorsements are of the form  $(AIEPubKey, blockNumber, sign)$ . An endorsement is verified by reconstructing the signed message (see *endorseAttributes*), hashing it, and verifying the signature.

### 5.6.3 Trust Level Computation

The trust level of an attribute represents how much the device can trust that a requestor actually possesses this attribute. It is a numerical value comprised between 0 and 1. 0 corresponds to a situation where the device cannot conclude with any degree of certainty whether the requestor possesses said attribute. It is used as a parameter in access policies. A low trust level may be enough to grant access to a less critical

resource while the main functionality of a device requires a trust level close or even equal to 1 to access. Administrators are free to fix their own requirements in this matter.

If no endorsement exists for a given attribute, the trust level is set to 0 and the request is most likely denied. In other cases, two methods of computation can be used.

**Trusted entities** Administrators can declare AIE that should be considered as trusted entities (TE). Such a declaration can occur at different levels. It can be policy-specific, attribute-specific, or general.

As seen in Section 5.4, a policy may specify an AIE. This AIE is then considered as a trusted entity with regards to that policy. Administrators can also decide to trust an AIE with a number of specific attributes. An example could be a company using its own AIE to issue roles within its organization. In that case and for all policies, the *role* attribute would need to have been endorsed by their in-house AIE to be deemed valid. This is materialized by the optional scope parameter in the *newTrustedEntity* function of TrAnC. Finally, administrators can declare general TE that are valid across policies and attributes. All the above definition of TE stack and an AIE can be considered as trusted thanks to different attributions.

An attribute endorsed by a TE is given a trust level of 1. It is a matter of policy whether this trust level is a requirement. Administrators are still free to require a lower trust level for a given attribute and a given policy.

When a device retrieves the list of endorsements for an attribute, it looks for potential trusted endorsements. If any is found and successfully verified, the attribute is given a trust level of 1. If no TE are declared or none have endorsed the attribute, the device must use the reputation system to compute the attribute's trust level.

**Reputation System** In the absence of trusted entities, a reputation system is used to determine the trustworthiness of an endorsement. This reputation system can reside on the blockchain [Dennis and Owen, 2015, Schaub et al., 2016] or outside of it. The trust level of an endorsement is based off of its endorsing AIE's reputation, normalized to a number between 0 and 1. Additionally, the reputation scores retrieved via the *getReputationScores* function of a device's TrAnC are used to determine the trustworthiness of an endorsement. When defined, they take precedence over the reputation system. This enables the blacklisting of entities in response to an attack for instance.

The final trust level of an attribute is computed by combining individual endorsements' trust levels. The combination methods must be chosen in accordance to the use case and the environment. Examples of such methods include taking the maximum trust level, the minimum, the mean of all the endorsements, or even the trust score occurring the most often.

#### 5.6.4 Access Decision

Each attribute is considered valid if the computed trust level is greater than the minimum value set in the policy. If all required attributes are valid, access is granted. Otherwise, the request is denied.

## 5.7 Security Analysis

We show that, given the security assumptions described in Section 5.2.2, our system is secure against the following attacks, assuming that the device is physically tamper-proof.  $\mathcal{A}$  denotes the attacker.  $\mathcal{H}$  is an honest participant.

### 5.7.1 Identity Theft

$\mathcal{A}$ 's goal is to masquerade as the honest requestor  $\mathcal{H}$ , thus enjoying the privileges offered by  $\mathcal{H}$ 's attributes. In order to do that,  $\mathcal{A}$  must be able to:

1. forge a valid signature of  $\mathcal{H}$  within the access protocol (step 1 of Figure 5.2),
2. replay one of  $\mathcal{H}$ 's signature from an old protocol session.

Forging a valid signature not knowing the private key of  $\mathcal{H}$ , breaks assumptions  $A1$ , or  $A2$ .

Replaying  $\mathcal{H}$ 's signature means that the random generator used by the device is not robust, which contradicts assumption  $A7$ .

### 5.7.2 False attributes endorsements

$\mathcal{A}$ 's goal is to convince the device that she has attribute  $a$ . To achieve this, she can:

1. declare herself as an administrator to promote a fake AIE and issue her own endorsement,
2. compromise a trusted AIE and have it endorse  $a$ ,
3. inject another AIE's signature for  $a$  in step 3 of Figure 5.7,
4. replay a valid endorsement transaction previously performed by an AIE,
5. game the reputation system to inflate the reputation score of her own AIE and have it endorse  $a$ .

Option (1) breaks assumptions  $A1$  or  $A2$  as invocation of the *newAdmin* or *newTrustedEntity* functions of the TrAnC requires administrator's credentials. Another road would be to collude with an administrator which breaks assumption  $A5$ .

Option (2) similarly breaks assumptions  $A1$ ,  $A2$ , or  $A5$ . Even if an AIE is compromised, as long as their private key is safe ( $A1$ ),  $\mathcal{A}$  cannot forge their endorsement as this would break assumption  $A2$ . Collusion breaks assumption  $A5$ .

Option (3) requires  $\mathcal{A}$  to forge a signature over  $a$ , which contradicts assumptions  $A1$  or  $A2$ .

Option (4) is detailed in section 5.7.4.

Option (5) contradicts assumption  $A8$ .

### 5.7.3 Privilege Elevation

$\mathcal{A}$ 's goal is to convince the device that she is allowed to access resource  $r$  controlled by policy  $P$ . She would need to:

1. produce false attribute endorsements corresponding to  $P$ 's requirements,
2. change  $r$ 's policy assignment in its DisC to point to a policy that  $\mathcal{A}$  fulfills,
3. update  $P$  in the PoIC to alter either the trust level associated to each attribute or the attributes themselves.
4. substitute herself to the gateway to replace  $P$  by a policy that  $\mathcal{A}$  fulfills,
5. compromise the gateway to replace  $P$  by a policy that  $\mathcal{A}$  fulfills.

Option (1) has been treated in section 5.7.2.

Option (2) breaks assumptions  $A1$  or  $A2$  as invocation of the *addAssignment* or *updateAssignment* functions of a DisC requires administrator's credentials.

Option (3) breaks assumptions  $A1$  or  $A2$  as invocation of the *updatePolicy* function of a PoIC requires administrator's credentials.

The success of option (4) depends on the security of the device/gateway pairing. This process is considered out of scope and therefore cannot be analyzed here. However, if the substitution can be achieved,  $\mathcal{A}$  controls the update that are sent to the device. This includes policies and AIE's trust scores.  $\mathcal{A}$  can therefore gain access to  $r$  illicitly.

Option (5) is similar in outcome to option (4) if the gateway is compromised.

### 5.7.4 Replay attacks over endorsement transactions

$\mathcal{A}$ 's goal is to convince the device that she has attribute  $a$  by using previously issued transactions or any message exchanged between actors.

The payload of an endorsement transaction contains a signature over the following message:

$$id||attr||blockNumber$$

Unless she can forge an AIE's signature, thus breaking assumption  $A1$  or  $A2$ ,  $\mathcal{A}$  cannot reuse transactions where the  $id$  is not hers. Similarly, she cannot substitute one attribute for another.

To avoid the risk that a revoked attribute is successfully re-issued to the same requestor at a later time without the initial AIE's approval, AttC should be parameterized to only accept attribute endorsements if the included block number is within an acceptable window around the current block number.

The blockchain does not offer any guarantees regarding the delay between the publication of a transaction and its validation and subsequent integration into a block. However, an average validation time can be approximated and used to define an acceptable window. It should be smaller than the life expectancy of attributes. Indeed, if *blockNumber* is still within the acceptable window when the attribute is revoked, the payload of the initial transaction can simply be re-used.

The inclusion of a challenge between the requestor and the device prevents  $\mathcal{A}$  from replaying old access request sessions.

### 5.7.5 Replay attacks over policy transactions

$\mathcal{A}$ 's goal is to convince the device that access to  $r$  is protected by  $P$  by using previously issued transactions or any message exchanged between actors.

Contrary to attributes that can be endorsed by any entity, policies must be issued or revoked by administrators. Extracting the payload of a previous transaction is therefore not enough. A new transaction must be issued and must contain the signature of an administrator to be considered valid by the blockchain. Thus, transactions cannot be re-used to update policies.

### 5.7.6 Privilege suppression

$\mathcal{A}$ 's goal is to wrongly convince the device that  $\mathcal{H}$  does not possess attribute  $a$ . This might be done by:

1. using a compromised AIE to revoke attribute  $a$ ,
2. compromising administrators and revoking the AIEs having endorsed  $\mathcal{H}$  with  $a$ ,
3. replaying the revocation transactions for  $a$ ,
4. altering the blockchain content thus removing  $a$  for  $\mathcal{H}$ ,
5. stopping the transaction endorsing  $a$  for  $\mathcal{H}$ ,
6. removing the AIE's signature for  $a$  in the step 3 flow of Figure 5.7,
7. blocking the communication between the device and its gateway.

Options (1) and (2) contradict assumptions  $A5$ ,  $A1$  or  $A2$  (similarly explained in section 5.7.3).

Option (3) assumes  $a$  has been revoked in the past, and  $\mathcal{A}$  tries to replay the revocation transaction. The block number included belongs to an older transaction and will not match the one currently stored.

Option (4) contradicts the blockchain consistency assumption ( $A3$ ).

Option (5) contradicts  $A4$  as  $\mathcal{A}$  can not stop endorsement transactions from being validated.

Option (6) and (7) are dependent on the local protocol used between the gateway and the device. No protocol can mitigate a network shutdown. They remain small scale attacks occurring at the local level.

### 5.7.7 Other issues and mitigation

**Gateways** Overall, as with many IoT application, the gateway represents a weak point in our system. Through it, an attacker can modify policies and trust scores given to the device. This attack however only has local consequences as only a few devices depend on any given gateway. This is a betterment compared to a centralized architecture. To mitigate its effect, the device can be either directly connected to the blockchain, if it has enough capacities, or connect to more than one gateway and compare information.

Alternatively, updates pertaining to policies or trust score can include the corresponding blockchain transactions. This offers two options. The first option is for the device to contact any other blockchain node to verify the transactions' legitimacy.

The second option is to maintain a list of administrators and verify that the transactions have been issued by one of them. This requires the device to be configured with at least one administrator's public key during the bootstrap phase. The list of administrators can still be extended later on: a call to the *newAdmin* function coming from the original administrator will be considered as valid, the new administrator will therefore be dynamically included in the device configuration. In this second option, devices should be wary of unconfirmed transactions. A malicious gateway could indeed update their devices and produce a valid transaction that is yet to be included in the blockchain. This opens the door for *double spending*. The consequences of double spending in this context however are limited.

**Smart Contracts** The implementation of the different smart contracts has direct implications on the security of the system. As we have seen in Section 2.8.5, bugs in smart contracts can be exploited to devastating effects. The risk is even greater as users deploy their own contracts. A formally verified template should be provided that all actors can use and modify at their own risk.

When a vulnerability is found in a smart contract, it must be deleted and a patched version redeployed. For TrAnC and DisC this implies a change in the device configuration as the new contracts will not have the same blockchain address. DisC and PoIC can also be affected if their respective TrAnC is removed from the blockchain. PoIC and AttC can be redeployed without affecting the device. A PoIC redeployment however affects the DisC that references it.

**Denial of Service** We have seen that an adversary could perform privilege suppression attacks by filtering out endorsements transiting between the device and the gateway. Our system presents four communication segments and as many opportunities to block information during its transport. The segments are: the blockchain network, the communication channel between the blockchain and the gateway, the one between the gateway and the device, and finally the one between the device and the requestor. The disruption of communication does not have the same effects depending on the segment.

Blockchains present their own protection mechanisms against DoS attacks. They are implementation-dependent. Section 2.8.6 addressed this concern.

Depending on the quality of the gateway's network access, stopping communication between the blockchain and the gateway can be hard. If the quality is poor to begin with, then communications can be interrupted by targeting the gateway's access directly, cutting it from all communications. Otherwise, the blockchain is a peer to peer network. The gateway therefore has many options of peers to connect to. Blocking them all would be difficult. If the connection is interrupted nonetheless, the system will continue to operate with the parameters that were last received. Revocations and other updates will not go through. Once communication is restored, the gateway can catch up to the rest of the blockchain, update its devices, and resume as normal.

Blocking the short range communication between the requestor and the device will totally hinder operation. However, there are no protocol-level protection against that.

## 5.8 Comparison to the state of the art

Table 5.5 compares our solution to the state of the art.

We adopted a blockchain-based architecture akin to the one presented in Section 3.7.3. The use of the blockchain result in a few similarities: Policies are stored on the blockchain. Permission updates are therefore put into practice after being validated by miners and integrated in the blockchain. In our proposal, policies are also stored locally on the device. The solutions have very high resilience as they are based on a distributed peer to peer network. The blockchain of course provides auditability. Note that in our case, policy history can be traced back but access requests are not stored in the blockchain. Nodes can be updated and maintained without affecting operation. Note however than redeploying faulty smart contracts is a costly operation.

Beyond their architecture, these solutions share a resource-level granularity, and a multi-head governance.

Resource efficiency varies across solutions. We use a gateway to mitigate the high resource demand of the blockchain. The local evaluation of attribute trust level and storage of policies can still be demanding for constraint devices. Hence our average score in that category. Our analysis is not supported by an implementation that would let us quantify the impact of such operations on IoT devices.

We only provide explicit revocation. On the blockchain, there are no guarantees regarding integration time once a transaction has been issued. This means that revocation transactions take an unknown time to be validated. For certain use case, this is not acceptable. Our solution would therefore be greatly improved with the addition of implicit revocation. This does not require heavy modifications of our proposal.

As the majority of the state of the art, our solution does not address delegation.

Our solution shines by enabling serverless access, or rather local access through the gateway. Ouaddah et al. [Ouaddah et al., 2016a] for instance require users to contact the device owner and a transaction to be validated to grant access. Our solution is more flexible and scalable. Zhang et al. [Zhang et al., 2018b] use one contract per resource/subject pair and per authentication method, leading to static policies. By using generic policies and detailing how they can be managed and updated, we minimize the space taken by our access policies while maintaining expressivity. The use of attributes provides a higher level of context-awareness. The issuance and management of attributes is lacking in [Pinno et al., 2017]. Users are in control of their identities. This enhances privacy and usability.

## 5.9 Conclusion

This chapter describes a blockchain-based attribute and policy management system for access control in the IoT which is user-centric, easy to manage, flexible, and interoperable. This system benefits from blockchain properties such as its distributed nature, resilience, and auditability. The underlying shared architecture lowers the deployment cost for smaller device owner, thus making security more accessible. Clients can use several pseudonyms to split their attributes amongst several identities, enhancing their privacy.

The administrators of the IoT system are free to rely either on a small number of entities that they fully trust to issue attributes to requestors, or they can leverage a reputation system and agglomerate many entities's endorsements of an attribute for verification. The entire lifecycle of an attribute managed through the blockchain, including its revocation is detailed in our proposal. The issuance and revocation



Criteria	MAAC-B	[Zyskind et al., 2015]	[Ouaddah et al., 2016a], [Ouaddah et al., 2017a]	[Novo, 2018]	[Pinno et al., 2017]	[Xu et al., 2018a]	[Zhang et al., 2018b]
Model	ABAC	-	-	-	-	-	-
Crypto	-	encrypt: symmetric, sign: ECDSA	ECC	asymmetric	asymmetric	-	-
Standards	-	Kademilia, LevelDB	Bitcoin	CoAp, JSON-RPC	-	Ethereum, JSON, SQLite	Ethereum
Use Case	Smart Lock	Mobile user	-	-	-	-	-
AuthN or AuthZ	authN & authZ	authN & authZ	authN & authZ	authZ	authZ	authN & authZ	authZ
Policy storage	on BC, on Device	on BC	on BC	on BC	on BC	on coordinator, on Device	on BC
Bootstrap phase	no	yes	yes	yes	no	yes	yes
Resource Efficiency	fair	poor	poor	high	high	fair	fair
Resilience	very high	very high	very high	very high	very high	very high	very high
Serverless	local	no	yes	no	-	no	yes
Scalability	high	very poor	high	poor	very high	fair	fair
Maintainability	high	high	high	high	high	fair	high
Permission Updates	Int BC	Int BC	Int BC	Int BC	Int BC	Int BC	Int BC
Usability	high	high	fair	high	fair	very high	high
Granularity	RL	RL	RL	RL	RL	RL	RL
Context-awareness	high	very poor	high	very poor	very high	fair	fair
Revocation	expl	expl	expl	impl & expl	-	impl & expl	-
Delegation	no	no	yes	-	-	yes	-
Auditable	yes	yes	yes	yes	yes	yes	yes
Privacy	high	very high	fair	poor	poor	poor	poor
Governance	multi-head	multi-head	multi-head	multi-head	multi-head	multi-head	multi-head
Maturity Level	Th	Th	SE	SE	Th	SE	SE

“-”: Non Mentioned or Non Applicable - *RL*: Resource Level - *expl*: explicit - *impl*: implicit - *authN*: authentication - *authZ*: authorization - *LoT*: Lifetime of Token - *SE*: simulated environment - *LoS*: lifetime of a session - *Int BC*: integration in the blockchain - *Th*: Theoretical

Table 5.5: Comparison of MAAC-B with the state of the art

of access control policies is also presented.

Thanks to the blockchain’s decentralized nature, our approach supports dynamic administration. For instance, IoT devices might be dynamically updated with changes in an entity’s reputation and list of administrators. Policies can be updated.

We identify four directions in which this proposal can be extended.

1. **Implementation** - The system should be implemented to test its applicability.
2. **Accountability system** - To distribute trust further, the reputation system should be integrated to the access control system. Past behaviors can be recorded and attribute endorsements accepted based on the community feedback on access control.
3. **More context** - Attributes are only given to subjects. Our scheme can be extended to include environmental or object related attributes.
4. **Implicit revocation** - To mitigate the delay in revocation processing, attribute endorsement transaction might be enriched with an expiration time, thus allowing an attribute to be revoked without the need for a new blockchain transaction.



## Chapter 6

# IoT devices' lifecycle: ownership change and remote configuration

### Contents

---

<b>6.1</b>	<b>Introduction</b> . . . . .	<b>166</b>
<b>6.2</b>	<b>Security Considerations</b> . . . . .	<b>167</b>
6.2.1	Security assumptions . . . . .	167
6.2.2	Threat model . . . . .	168
<b>6.3</b>	<b>Asset ownership</b> . . . . .	<b>169</b>
6.3.1	Motivation . . . . .	169
6.3.2	Proposal . . . . .	171
6.3.3	Security analysis and limitations . . . . .	175
6.3.3.1	Blockchain-related threats and limitations . . . . .	175
6.3.3.2	Use-case-related threats and limitations . . . . .	175
<b>6.4</b>	<b>Managing secrets</b> . . . . .	<b>176</b>
6.4.1	Motivation . . . . .	176
6.4.2	Proposal . . . . .	177
6.4.3	Security analysis and limitations . . . . .	179
<b>6.5</b>	<b>Sharing configurations</b> . . . . .	<b>180</b>
6.5.1	Motivation . . . . .	180
6.5.2	Proposal . . . . .	181
6.5.3	Security analysis and limitations . . . . .	181
<b>6.6</b>	<b>Conclusion</b> . . . . .	<b>182</b>

---

## 6.1 Introduction

Up until now, this thesis has been focused on access control solutions. Chapter 5 explored the adequacy of a blockchain-based solution. In this chapter, we come back to a more traditional use of the technology, asset exchange, and use it to manage ownership transfers.

A shorter version of this proposal was presented at SecureComm 2018 [Dramé-Maigné et al., 2018]. The original paper focused on ownership tracking and presented secret management as an extension. The present chapter presents a second extension: dynamic configuration sharing.

As mentioned, a common blockchain use case is the tracking of asset ownership such as houses, cars, or artwork [Rosenfeld, 2012]. In some countries the state cannot be trusted to keep accurate records of land ownership due to corruption. The blockchain constitutes a viable trust-less alternative. Countries such as Georgia<sup>41</sup>, and Sweden<sup>42</sup> are each at different stages of implementing a land entitlement project using the blockchain. Such initiatives have the power to combat that corruption and give power back to farmers and small property owners.

In this chapter, we propose to use the blockchain to track IoT devices' ownership. The blockchain is a cheaper alternative to ownership records when compared to traditional methods that involve an outside authority such as notaries. It is also a simpler and faster process. Thanks to the decreased cost and added usability, ownership records can then be used for more low-cost assets such as IoT devices.

This mechanism can also be used to exchange device-related secrets, enabling remote configuration and efficient secret management. IoT use cases can involve many devices deployed in various physical locations. This makes manual configuration inefficient. Smart grids are a good example of hundreds of devices that need to be deployed to cover the entirety of electricity grids. The deployment speed is highly impacted by the configuration method, as many devices need to be configured at once. By leveraging the chain of ownership published in the blockchain, we propose to facilitate remote configuration. Additionally, owners can use the same mechanism to efficiently manage the multiple secrets used to remotely manage their devices.

The same mechanism can also be used to publish information related to IoT devices, either for safekeeping or for advertising their characteristics to potential users. IoT use cases depart from classical ones that involve only a small number of known actors. The list of clients for one IoT device can be dynamic. In order for that device to be trusted with a task, potential clients may require some guarantees. For instance, a user could demand that the device be running the latest version of a software, hence ensuring that security patches have been applied, or simply that its list of communication protocols intersect with its own. Such information can be published on the blockchain.

**Contributions** The contributions of this chapter can be summarized as follows:

- An independent proof of ownership based on blockchain transactions,
- The desintermediation and decentralization of ownership records,

---

<sup>41</sup>[https://www.mitpressjournals.org/doi/pdf/10.1162/inov\\_a\\_00276](https://www.mitpressjournals.org/doi/pdf/10.1162/inov_a_00276), Last checked July, 14th 2019

<sup>42</sup><https://www.coindesk.com/sweden-demos-live-land-registry-transaction-on-a-blockchain>, Last checked July, 14th 2019

- A key management system,
- The advertising of dynamic device properties to potential users.

Symbol	Description
$D$	Device
$\{D_i\}_{0 \leq i < n}$	Family of $n$ devices
$id_i$	Identifier of device $D_i$
$O$	Device owner
$M$	Device manufacturer
$C$	A Company
$addr_A$	Blockchain address of $A$ . $addr_A = Hash(pub_A)$
$(pub_A, priv_A)$	Public/private blockchain key pair linked to $addr_A$
$s_i$	Secret linked to device $D_i$
$K_A$	Master key of $A$
$k_{A,B}$	Symmetric key derived from $K_A$ and shared with $B$
$k_{A,i}$	Symmetric key derived from $K_A$ and $id_i$
$tx_k$	$k^{th}$ blockchain transaction
$out_j^k$	$j^{th}$ output of $tx_k$

Table 6.1: Chain of Ownership: Notations

**Organization** Notations are summarized in Table 6.1. The rest of this chapter is organized as follows.

Security assumptions and threat models are presented in Section 6.2. Section 6.3 introduces our tracking of ownership using the blockchain. Section 6.4 proposes an extension of the approach to configure IoT devices and manage keys for the sake of the owner. Section 6.5 discuss a second extension that advertises dynamic device properties to potential users. Finally, Section 6.6 concludes this chapter.

## 6.2 Security Considerations

### 6.2.1 Security assumptions

We operate under the following assumptions:

- A1 *Secured blockchain keys*: Blockchain keys cannot be stolen, lost or otherwise compromised. This implies good key management.
- A2 *Solid cryptographic primitives*: Our proposal uses cryptographic primitives such as signatures, hashes, or encryption. We assume these primitives cannot be broken.
- A3 *Blockchain consistency*: Fundamental blockchain properties include consistency amongst nodes and consistency over time [Pass et al., 2017]. This implies that all nodes in the network will

agree on blockchain history, the few last blocks excluded, and that accepted transactions cannot be modified. We assume these properties are verified and the blockchain history cannot be altered.

*A4 Blockchain capability:* All actors ( $M$ ,  $C$ ,  $O$ , and  $U$ ) own a blockchain address, the corresponding public and private key pair, and the means of submitting or retrieving a transaction to or from the blockchain.

*A5 Reputation System:* The actors take part in a reputation system where bad behaviors can be reported. We assume this system cannot be tampered with.

*A6 Unicity of device's serial number:* A device can be uniquely identified by its serial number. We assume this serial number is physically located on the device and cannot be tampered with.

*A7 Secure random generator:* Actors have access to a secure source of randomness.

A1 is a very strong assumption that does not really hold. However, as this is a core issue for all blockchain applications, there exist a number of methods to safeguard one's key. This issue has been addressed in Section 2.8.2.

For the same reasons, we do not address the security of reputation systems (Assumption A5).

## 6.2.2 Threat model

Across our three proposals, we consider three types of attackers : a malicious new owner, a malicious previous owner, and a malicious uninvolved third party. We detail nine possible threats involving these actors. These threats are summarized in Table 6.2.

**Malicious previous owners** This attacker's goal is to either fool a potential buyer, by not providing the device after the sale has been concluded, or to retain access to said device and thus gain access to sensitive data belonging to the new owner. As the previous owner, the attacker is in possession of the credentials that, at the time of the handover, enable device access. She can also provision anything onto the device prior to the handover and is able to produce a valid proof of ownership.

When a device is sold and exchanged, the previous owner can use her knowledge to gain access to sensitive information. She can also use the device as an entry point into the new owner's network. This defines Threat  $T1$ .

A prospective owner can be fooled by the previous owner and buy a device that will not be delivered. This defines Threat  $T2$ .

When a secret must be provided in order to gain access to the device (see Section 6.2), the attacker may refuse to provide it or falsify it, thus preventing the new owner from accessing his device. This defines Threat  $T3$ .

**Malicious new owners** The goal of this attacker is to gain access to sensitive information without authorization. As its new owner, the attacker has full access and full control over the device.

After the sale, if the device has not been properly wiped, the new owner can extract potentially sensitive information related to the former owner from the device itself. This defines Threat  $T4$ .

Nbr	Attacker Type	Description
<i>T1</i>	Prev. Owner	Previous owner retains access to the device
<i>T2</i>	Prev. Owner	Proof of Ownership is produced but device is not provided
<i>T3</i>	Prev. Owner	Secret is not valid. Provided device cannot be accessed
<i>T4</i>	New Owner	New owner extracts sensitive data from the device
<i>T5</i>	New Owner	New owner uses device to gain access to sensitive data
<i>T6</i>	Third Party	Ownership transfer cannot be completed
<i>T7</i>	Third Party	Attacker accesses sensitive information by eavesdropping
<i>T8</i>	Third Party	Attacker successfully masquerades as the device owner
<i>T9</i>	Third Party	Ownership chain with no corresponding device

Table 6.2: Threats

The new owner can also use the device's identity to gain access to previous owner's data. This can be achieved by interacting with users or devices that still recognize the device as being owned by the previous owner. This defines Threat *T5*.

**Malicious third party** This attacker's goal is to appear as a legitimate device owner to fool a potential buyer, steal and re-sale a device, disturb the sale transaction or gain information about the parties involved in the ownership transfer. When a public blockchain is used, the attacker has access to all information that transits through the blockchain. She can also produce and submit valid blockchain transactions.

First, the attacker can try to clog the blockchain network. In this event, the network would not be able to process the transaction signaling the ownership transfer. This defines Threat *T6*.

Second, when the transfer occurs, the attacker may try to gain knowledge about the involved parties. This defines Threat *T7*.

Third, the attacker may pretend to be the owner of a device she does not possess or acquired illegally (through theft for instance). This is Threat *T8*.

Fourth, the attacker may fabricate a blockchain trace for a device that does not actually exist. This is Threat *T9*.

## 6.3 Asset ownership

### 6.3.1 Motivation

As previously mentioned, asset tracking is one of the most straightforward blockchain application. Assets that have been considered for this use case tend to be expensive (i.e. land, cars, houses, paintings, etc). These objects' ownership will most likely already be tracked using third parties such as notaries, insurance companies, or other government-sanctioned entities. The corresponding administrative procedures can be long and costly. By using the blockchain instead, trust in these third parties and their infrastructure is no longer required. The cost of a transaction is also highly reduced. The transfer of ownership is a simple blockchain operation. For these reasons, ownership records do not have to be confined to expensive items. We propose to apply this principle to IoT devices.

Triggers for ownership transfer in the IoT might be: a user re-selling an old device after it has been replaced by a newer one, an individual looking for cheaper options and turning to the second-hand market, a company re-assigning resources as a project closes, long-term renting of IoT devices (for smart buildings applications for instance), etc.

**Benefits of the proposal** Keeping ownership records on the blockchain offers the following benefits:

- **Desintermediation:** Traditionally, changes in ownership must be attested, assisted, and recorded by third parties. As the blockchain keeps a public proof of the transaction, they are no longer necessary.
  - *Lower Cost* - Third parties involved in ownership transfers take a commission that drives up the cost of the operation. Desintermediation therefore has the added benefit of lowering costs.
  - *Freedom of choice* - Some users may prefer to still use a third party as a buffer between the vendor and the buyer and as a mean of solving potential conflicts. Instead of selecting this third party from a small pool of government-sanctioned agents, users are free to choose anyone that both party agree on.
  - *Independent Proof of Ownership (IPoO):* Proofs of ownership traditionally require an attestation from the third party in charge of the record. This requires the user to register and sometimes maintain an account with this entity. IPoO can be used to replace company-dependent registration processes, enabling a user to pseudonymously prove ownership of a device before maintenance operation or to obtain help from customer support.
- **Decentralization:** Records are neither controlled nor managed by a unique entity.
  - *Availability* - By its distributed nature, the blockchain offers availability guarantees that the deployment of a private fact recording infrastructure cannot match.
  - *Persistence* - Blockchain transactions will be stored in a decentralized fashion, protecting ownership record from loss and modifications by any one party.
- **Transparency:** Ownership records are publicly accessible. This gives a new buyer information about the life of a device, its age, maybe what it was previously used for, etc. Transaction history can easily be checked for discrepancies.
  - *Traceability* - Ownership of an object can be traced from its original owner to the most current one or the other way around. This enables application such as security alerts: When an incident affecting a large number of devices occurs, it is currently hard to track owners and warn them of the issue. Owners can be private individuals. They are not likely to follow best security practices. For that reason, in the event of a large scale IoT attack, being able to track and warn device owners could prevent further damage.



Field	Description	Status
Tx type	Possible values are <i>genesis</i> and <i>transfer</i>	<i>Mand</i>
Nounce	Can be made mandatory for <i>genesis</i> tx (see Section 6.3.3.1)	<i>Opt</i>
Inputs	Lists all the inputs of the tx (see Table 6.4)	<i>Opt</i>
Outputs	Lists all the outputs of the tx (see Table 6.5)	<i>Mand</i>

Table 6.3: Transaction format

- *Automation* - The availability of ownership transfer operation enables the automation of some processes that should occur when a device changes hands. For instance, the automatic revocation of allotted permissions can be triggered. This would enhance the protection of former owners' private information.
- **Usability:** This method of ownership tracking is simple to grasp and deploy.
  - *Simplicity* - The process by which the ownership is transferred requires a single transaction. Its simplicity makes it highly usable, even to private individuals.
  - *Interoperability* - When using existing public or private blockchains, one can take advantage of the infrastructure already deployed by others. This use case does not require the deployment of a dedicated infrastructure nor the federation of a large number of systems to enable interoperability.
- **Privacy:** The use of the blockchain empowers users to decide who they can trust, who they want to interact with, and what information they are willing to share rather than having them forced into a pre-established system that may not suit them.
  - *Pseudonymity* - Traditionally, ownership records are nominative. This is natural as the PoO is linked to one's identity. When using the blockchain, device ownership is tied to the ownership of the corresponding blockchain private key. This enables the use of pseudonyms.
  - *Identity ownership* - Blockchain accounts are both created and controlled by users.

### 6.3.2 Proposal

We take our example at the very beginning of the ownership chain with the sale of a device  $D$ . We consider the following actors : the device's manufacturer  $M$ , and a company  $C$  that wishes to acquire  $D$ . Both  $M$  and  $C$  possess a blockchain address, the corresponding public and private key pair, and the means of submitting or retrieving a transaction to or from the blockchain.

**Transactions** The general idea is to link the asset's exchange to a series of blockchain transactions, thus creating a chain of ownership. There are two types of transaction available. The transaction that creates the link between the asset and its digital counterpart is the *asset genesis transaction*. Transactions that mark a change in ownership are *transfer* transactions. Transactions follow the Bitcoin model of input/output (see Section 2.5.1.1), meaning that each transaction uses previous transaction outputs as

Field	Description	Status
Previous tx	Hash of a previous tx	Mand
Index	Index of output to be used in previous tx, must be unspent	Mand
Public key	Public key that matches the address that owns the selected output	Mand
Signature	Must be signed with the priv. key that matches the given pub. key	Mand

Table 6.4: Input format

Field	Description	Status
Destination	Blockchain address of the output owner	Mand
Id	Device serial number, mandatory for <i>genesis</i> tx	Opt
Secret	See Section 6.4	Opt
Data	See Section 6.5	Opt

Table 6.5: Output format

inputs. Transactions are detailed in Table 6.3. Table 6.4 and Table 6.5 breakdown the construction of inputs and outputs respectively.

For an input to be valid, it must be signed with the private key corresponding to the output's destination address. Figure 6.1 provides an example: transaction  $tx_0$  has 2 outputs,  $out_0^0$  sent to  $addr_A$  and  $out_1^0$  sent to  $addr_B$ . Transaction  $tx_1$  uses  $out_0^0$  as input. To be valid, the input must carry the public key corresponding to  $addr_A$  along with a valid signature produced using  $priv_A$ , private key corresponding to  $addr_A$ . Because outputs only carry blockchain addresses, and because hashes are irreversible, the public key  $pub_A$  is needed for the signature validation (reminder:  $addr_A = Hash(pub_A)$ ). Each output in a transaction corresponds to a different asset.

A *genesis*-type transaction has no input. Its outputs however must include an *id* field. According to A6, device identifiers are unique and cannot be tampered with. This field, shown in Table 6.5, therefore strongly affiliates a physical IoT device and its digital counterpart.

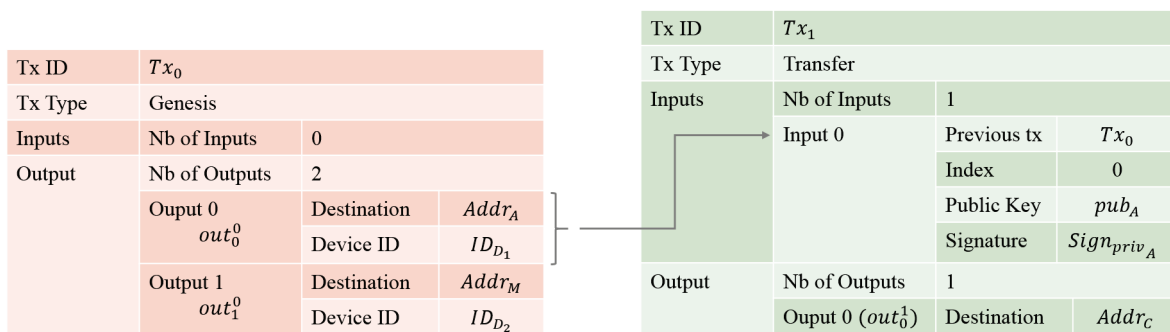


Figure 6.1: An example of blockchain transactions

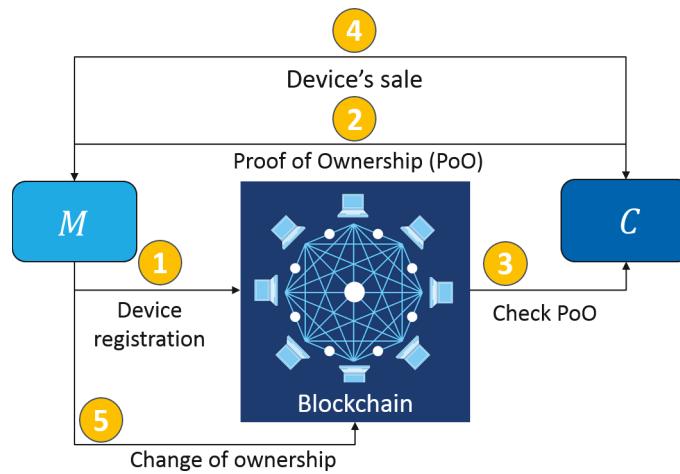


Figure 6.2: Overview of ownership transfer

**Ownership transfer** Figure 6.2 presents an overview of the steps required to transfer the ownership of device  $D$  from  $M$  to  $C$ . First,  $M$  must register  $D$  in the blockchain (step 1). They do so by issuing a *genesis* transaction. That transaction must contain an output addressed to  $addr_M$ , their own blockchain address, carrying  $D$ 's serial number. This creates device  $D$ 's digital representation and registers  $M$  as the original owner.

$C$  requires a Proof of Ownership before they can commit to purchasing  $D$ . The specifics of this proof are addressed in the next section. It involves an exchange of messages between  $M$  and  $C$  (step 2), as well as the retrieval of the *genesis* transaction from the blockchain (step 3). Once ownership has been established, the sale can proceed.

When  $C$  purchases the device,  $M$  issues a second transaction of type *transfer*. This transaction takes the previous *genesis* transaction as an input. It includes  $M$ 's signature and public key. The transaction also contains one output sent to  $addr_C$ . This second transaction transfers the ownership of  $D$  to  $C$ .

**Proof of Ownership** Before the sale can take place,  $M$  must produce a valid proof of ownership to  $C$ . Figure 6.3 illustrates this process.

First,  $C$  sends a challenge message  $m$  to  $M$  (step 1). The challenge is chosen by  $C$ .  $M$  therefore cannot reproduce an intercepted message.  $M$  signs  $m$  with  $priv_M$  (step 2) and sends back the proof of ownership itself (step 3). It must contain four things:

1. The identifier of the transaction that transferred ownership of the device to  $M$  ( $Tx_i$ ).
2. The output number corresponding to the device in question ( $j$ ). This output must be unspent.
3. The computed challenge response ( $s$ ),
4. The public key that goes with the transaction output ( $pub_M$ ).

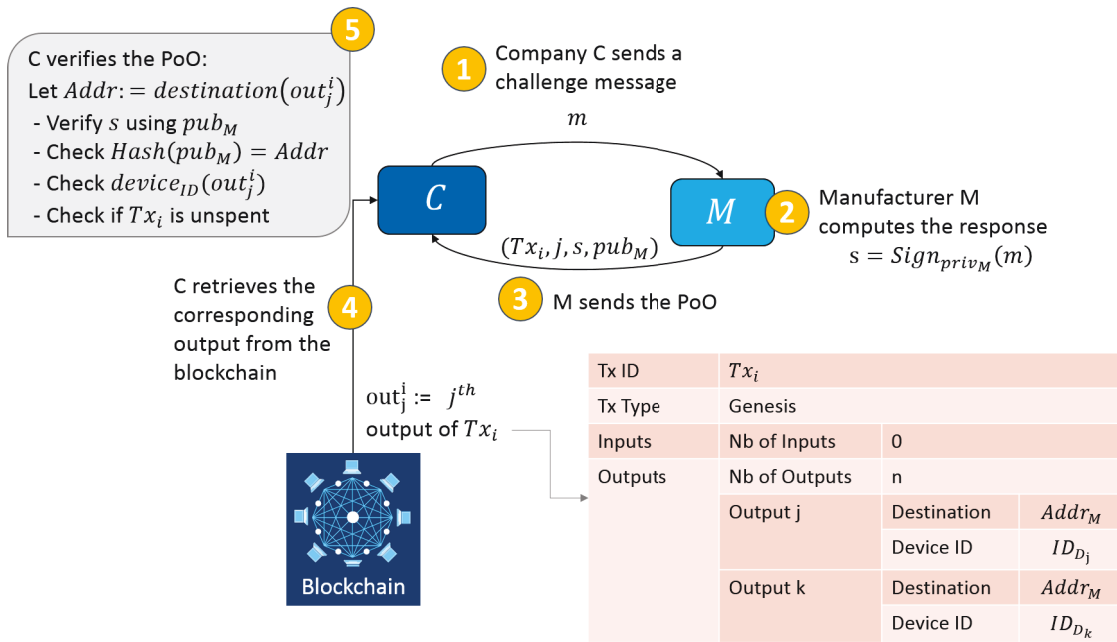


Figure 6.3: Proof of ownership

Using the transaction identifier,  $Tx_i$ , Company  $C$  can extract the corresponding transaction in the blockchain and in particular the designated output, here the  $j^{th}$  (step 4). To verify the PoO,  $C$ :

- Verifies the validity of  $M$ 's signature over the challenge message  $m$ . This ensures that  $M$  owns  $(pub_M, priv_M)$  and by extension  $addr_M$ .
- Extracts the destination address from  $out_j^i$  and compare it to the hash of the public key provided in the PoO.
- Checks that the device ID registered in the transaction correspond to  $D$ .
- Validates that  $Tx_i$  is unspent.

Note that providing such a proof does not compromise the owner's pseudonymity.

Following Assumption  $A1$ , blockchain keys cannot be stolen. Assumption  $A2$  states that the blockchain's cryptographic primitives cannot be broken. This means that the only person capable of producing a valid PoO is the owner of both the blockchain key and the device. Furthermore, since the blockchain history cannot be altered according to Assumption  $A3$ , once an ownership record has been published or updated, it cannot be modified. This neutralizes Threat  $T8$ .

A PoO can be required for operations outside of the scope of a sale. Currently, when buying a device, the customer can be required to register with the corresponding vendor. This involves personal data that will later be used to authenticate and identify the customer as the device owner. A PoO can be used instead, enhancing customer privacy.

### 6.3.3 Security analysis and limitations

Our proposal presents a few limitations that we describe below. Some are due to the underlying blockchain. Others stem from the use case itself. We also discuss how our scheme mitigates the threats defined in Section 6.2.2.

#### 6.3.3.1 Blockchain-related threats and limitations

The principal limitation of this solution is that the security of the scheme depends on the security of the underlying blockchain [Lin and Liao, 2017]. These concerns have been expressed in Sections 2.8 and 2.9. Assumption A2 does not cover these issues as they are not crypto-related but rather network-related. However, despite all these theoretical shortcomings, blockchains like Bitcoin and Ethereum have demonstrated their resilience to attacks and only grown stronger as a result.

Another issue that needs addressing is the resistance to DDoS attacks. In the Bitcoin blockchain, the only transactions without inputs are *coinbase* transactions. First transaction of a block, a *coinbase* transaction can only be issued when a block is mined. Furthermore, now that Bitcoin miners' payment is moving from block reward to transaction fees, all other transactions have a cost that is only going to increase over time. This mitigates DDoS attacks as the cost is linear in the number of transactions. When a large number of transactions floods the network, miners can temporarily increase transaction fees, thus rendering an attack even more costly. If they do not require inputs, *genesis* transactions are not exempt from fees. Issuing a large number of them has a cost that is at least linear in the number of transaction and can even grow faster as the miners' fees adapt to the situation. The cost of the attack is a deterrent. This addresses Threat T6. Valid transactions can also be created by transferring a device's ownership to oneself. But the cost is the same.

We propose two additional means of mitigation. The first solution is to use a private blockchain where the right to issue *genesis* transactions is limited to pre-approved actors. Manufacturers would need to be registered in a manufacturer consortium, granting them the exclusive right of issuing *genesis* transaction, thus creating new devices. A manufacturer that behaves incorrectly, by advertising non existing devices or issuing too many *genesis* transactions, would lose its publication privileges. This has the added advantage of addressing Threat T9. But private blockchains unfortunately limit interoperability and do not offer the same decentralization-related properties when compared to public ones.

A second solution consists in increasing the cost of *genesis* transactions. They would require a *nounce* as an input (see Table 6.4). Similarly to Proof of Work (see Section 2.6.1, the *nounce* would be chosen so that the hash of the transaction is lower than a pre-defined threshold. The difficulty does not need to be as high as Bitcoin's PoW and can be adapted to counter DDoS attacks. The downside is that this increased computational cost will mostly impact manufacturers as they are the most likely to issue *genesis* transactions. This is therefore likely to impact the device's cost in return.

#### 6.3.3.2 Use-case-related threats and limitations

In the above proposal, a *genesis* transaction creates the digital representation of an IoT device. If the transaction is linked to the device via its serial number, no proof of the existence of this device is required. The production of a valid proof of ownership does not translate to the possession of a real-life IoT device.

In case of theft for instance, the original owner can still produce a valid proof but will not be able to produce the device itself. This situation is not different from online shopping where the buyer has to rely on pictures, listings, reputation, or other criteria to decide whether to trust the vendor. Following Assumption *A5*, vendors that do not provide devices after the transaction is completed can be reported. Their reputation score will be lowered and they are less likely to fool someone else in the future. This addresses Threat *T9*.

Similarly, the issuance of a *transfer* transaction does not force the shipping of the device to the new owner. It means however that the previous owner can no longer prove that they own the device. This is a deterrent as future prospective buyers are unlikely to commit to the sale if the ownership cannot be proven.

Blockchain transactions are irreversible. The vendor can therefore be tempted to require payment before the *transfer* transaction is issued. The buyer then runs the risk of that transaction never being issued. Bitcoin's multi-signature presents a solution to this problem. Multi-signature refers to transactions that need more than one signature to be valid. The desired number here is 2 out of 3. The buyer and vendor choose a party that they trust to be impartial. The buyer then sends the funds to the multisignature address. If everything goes smoothly, upon reception of the purchased item, the buyer and seller both sign the transaction and funds are sent to the vendor. When a conflict occurs, the third party decides who should receive the funds and signs the transaction together with the interested party. The same can be done with ownership transactions. This addresses Threat *T2*.

An owner could also try to sell the same device to two different people. This is a problem that is similar to double spending (see Section 2.14). In a similar fashion, both transactions cannot co-exist. New owners should therefore be sure to wait for the blockchain transaction to be confirmed. For Bitcoin, the generic rule is to wait for the transaction to be buried under 5 to 6 blocks, which takes around an hour. For such a use case, this delay is not an inconvenience. In all of the above cases, bad behavior from any of the involved actors will negatively affect that actor's reputation score (Assumption *A5*). All reporting should include the incriminating transaction(s) when applicable. For a double sale for instance, two transactions spending the same output with valid signatures should be provided as proof of bad behavior.

Finally, malicious previous owners might want to retain control of their former device after it has been shipped to its new owner. To protect against this risk, the device should be wiped clean upon reception and all the credentials should be changed. This addresses Threat *T1*. The same applies to a former device owner who wants to prevent her sensitive data from being accessed by the new owner. Before the device can be shipped, it should be restored to factory default. This addresses Threat *T4*. The necessary steps should also be taken to revoke the device's access to all sensitive services such as a smart home private network. This addresses Threat *T5*. Threats *T4* and *T5* are better addressed by Section 6.5.

## 6.4 Managing secrets

### 6.4.1 Motivation

Security rests on the sharing of secrets. These secrets are used to secure communications or encrypt data. When a device is manufactured, initial secrets are provisioned to start the security chain. When acquiring

a device, its secrets need to be retrieved from the manufacturer or previous owner. The means currently at our disposal to do so lead to slow and cumbersome deployment processes. What is needed therefore is a mean of efficiently retrieving that information to be able to remotely and efficiently configure devices in an industrial context.

Currently, physical access to the device is often necessary. When buying a device, the new owner will have it shipped to her location and configure it. The pin or the password may be written down on the device's box, in the configuration manual or otherwise physically attached to the device and its packaging. Buyer and seller might also choose to call on a trusted third party to take care of the configuration and installation of devices.

The need for an initial physical access is a hindrance on the deployment process. Because many devices need to be configured at once, this method that is slow, costly and may require to trust confidential information to a third party is ill-fitted.

Another issue is the management of these secrets. In IoT scenarios, multiple devices may be owned by the same entity. Furthermore, symmetric cryptography is often preferred due to the constrained nature of IoT devices. This is another multiplying factor for the number of keys involved. This multiplicity implies the need for an efficient management of secrets over the life a device. Based on the blockchain ownership records, we propose a solution that both delivers a device's secret to its newest owner and enable their management over the life of the device.

As an extension, this proposal presents the same benefits as those described in Section 6.3.1. To these we add the a cost reduction and simplification of the deployment process, the reduction of the number of secret keys, and a distributed storage for said keys.

## 6.4.2 Proposal

For the sake of this proposal, we consider IoT devices as black boxes exposing a number of functions that can be activated either by physical interactions or via a communication channel. In both cases, a secret is required to successfully invoke any function. When the device is manufactured, an initial secret is provided. As for any function, the generation of a new secret requires the previous secret and can be invoked either by physical interaction or through the communication channel.

Once again, we start at the beginning of the ownership chain. The manufacturer  $M$  sells a batch of  $n$  devices  $\{D_i\}_{0 \leq i < n}$  to a company  $C$ . Each device has a unique identifier  $id_i$ . Additionally,  $C$  owns a master key  $K_C$ . Used as an input for key derivation,  $K_C$  should not be shared and only be known by  $C$ . The symmetric key  $k_{C,M}$  is derived from  $K_C$  and  $M$ . This means that a symmetric key is associated with every vendor. Key  $k_{C,M}$  will be used to encrypt  $\{s_i\}_{0 \leq i < n}$ , secret linked to device  $\{D_i\}_{0 \leq i < n}$ . The blockchain still supports two types of transaction, *genesis* and *transfer*.

**Delivering device secret** Figure 6.4 illustrates the process by which ownership is transferred and secrets are exchanged :

Step 1  $M$  interacts with each  $D_i$  and generates a secret  $s_i$ . This secret  $s_i$  can be an administrative password, a private key, a pin, etc.

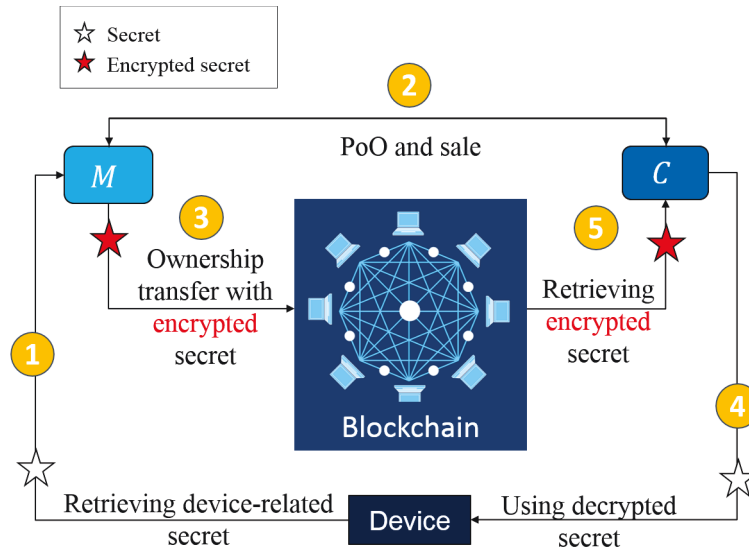


Figure 6.4: Transferring ownership and delivering device secret

Step 2  $M$  retrieves  $k_{C,M}$  from  $C$  or from its own record. That information can be provided along with payment information for instance. We assume  $k_{C,M}$  to be a symmetric key that needs to be provided by the buyer. If  $C$  prefers using asymmetric cryptography, the key used to encrypt  $s_i$  can also be retrieved from a registry storing public key records. These keys are used for application purposes and should differ from the keys used for the blockchain protocol. Using  $k_{C,M}$ ,  $M$  encrypts each  $s_i$ .

Step 3  $M$  issues a *genesis* transaction,  $tx_0$ , with  $n$  outputs where  $out_i^0$  is linked to  $D_i$  through its serial number  $id_i$  and is sent to  $addr_M$ , her own blockchain address.  $M$  issues a second transaction of type *transfer*,  $tx_1$ , with  $\{out_i^0\}$  as inputs, signed with  $priv_M$ . This transaction yields  $n$  outputs, one for each  $D_i$ , sent to  $addr_C$ . In addition to  $addr_C$ , each output carries  $Enc_{k_{C,M}}(s_i)$  (see Table 6.5).

Step 4  $C$  retrieves  $\{Enc_{k_{C,M}}(s_i)\}$  from the blockchain and deciphers them, recovering  $\{s_i\}$ .

Step 5 Using  $s_i$ ,  $C$  gains access to each  $D_i$ . When necessary,  $s_i$  is also used for configuration.

The same process can then be repeated by the new owner to sell the device to somebody else.  $tx_1$ 's outputs can be separated, enabling devices to be sold separately.

This scheme involves several keys and secrets, especially when considering devices bought from multiple vendors but only  $K_C$  and  $priv_C$  need to be safeguarded by  $C$ . Each  $s_i$  can be recovered from  $K_C$ . This greatly simplifies the management of secrets where many devices are involved.



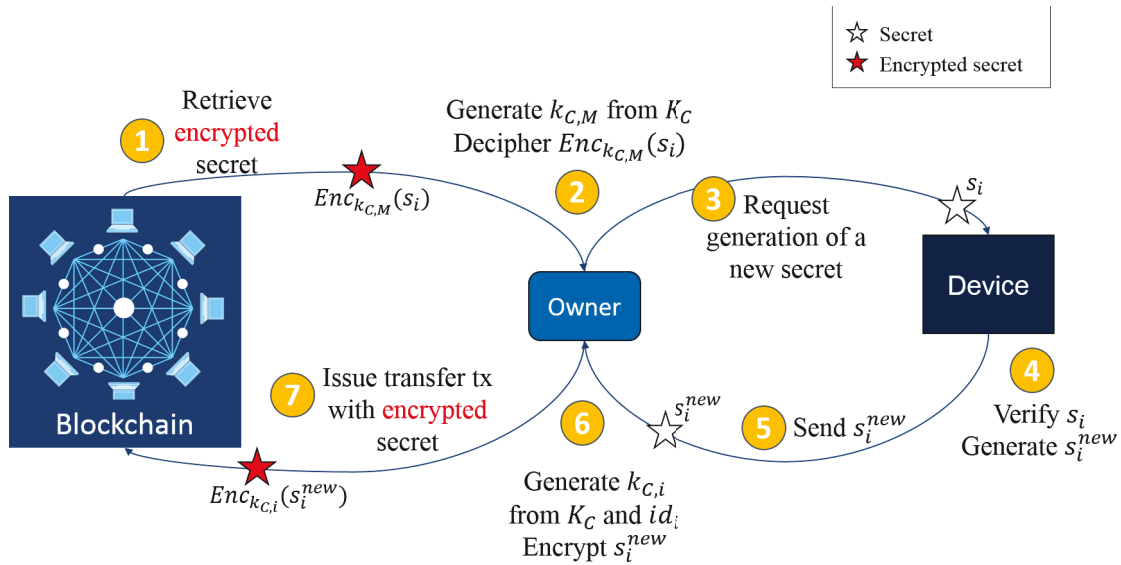


Figure 6.5: Publishing a new secret to the blockchain

**Updating device secrets** Updates can be made to a device’s secret, as illustrated by Figure 6.5. After buying a device  $D_i$ , the new owner should change the corresponding  $s_i$  as this secret is known to the previous owner (Threat  $T1$ ). The first step is to recover the encrypted secret from the *transfer* transaction published in the blockchain (step 1). Assuming that  $M$  is our vendor and  $C$  is the new owner,  $s_i$  is encrypted with  $k_{C,M}$ , the symmetric key shared by  $C$  and  $M$ . That key can be generated from  $K_C$  and used to decipher  $s_i$  (step 2).

Using  $s_i$ , the owner can invoke any function of  $D_i$  including the generation a new secret,  $s_i^{new}$  (step 3). If  $s_i$  is valid, the device generates  $s_i^{new}$  (step 4), and sends it back to the owner (step 5). This secret can now replace  $s_i$  in the blockchain.

The owner generates a new key  $k_{C,i}$  using their master key  $K_C$  and the device identifier  $id_i$ , and encrypts  $s_i^{new}$  (step 6). The last step is a simple *transfer* transaction to themselves, replacing  $Enc_{k_{C,M}}(s_i)$  by  $Enc_{k_{C,i}}(s_i^{new})$ , where  $Enc$  is the encryption algorithm (step 7).

Such a transaction can also be made to hide the link between the owner’s identity and their blockchain address, thus hiding the number of device belonging to a single owner. This also hides how long an actor holds on to a device (Threat  $T7$ ). Note that the new secret should of course be encrypted with a key that is not known to  $M$ . Here  $k_{C,i}$  is used to replace  $k_{C,M}$ .

### 6.4.3 Security analysis and limitations

The first delicate point of this scheme is the transmission of the encryption key,  $k_{C,M}$ , from  $C$  to  $M$ . If symmetric keys are used, then a secure communication channel should be put in place to enable the exchange. The security of a key during its transmission falls outside of the scope of this proposal. When

a large number of sales involve the same actors, here  $M$  and  $C$ , the same key is used to encrypt all secrets. The symmetric key must be exchanged only once. Alternatively, a master key can be used from which session keys are extracted for encryption.

If asymmetric cryptography is used, the public key can simply be transferred or even made available in a registry. Registries can be switched without affecting the scheme. This mitigates Threat  $T7$ .

Before selling a device, the owner should invoke the secret generation function to bind the device to a new secret. If the secret that is communicated to the new owner via the *transfer* transaction was already in use, it can be used to retrieve private information from the previous owner (Threat  $T4$ ). Similarly, if  $k_{C,M}$  is a symmetric key, it should not be reused when a new secret is uploaded to the blockchain. Otherwise, all  $s_i^{new}$  are exposed to  $M$  (Threat  $T1$ ). To achieve this,  $k_{C,i}$  can be derived from elements linked to the blockchain transactions. Let  $tx_n$  be the latest transaction that proves  $M$  owns  $D_i$ . Such a transaction must exist with unspent output otherwise  $M$  is not the rightful owner of  $D_i$ . In that case,  $Hash(tx_n)$  could be used as an input for key derivation. The *transfer* transaction from  $M$  to  $C$ , transaction  $tx_m$ , will use one of  $tx_n$  output as input. Similarly, when updating  $s_i$ , an output from  $tx_m$  will be used. The hash of the previous transaction is therefore an easy element to recover. It varies with transactions, leading to different  $k_{C,i}$ .

As stated in Threat  $T3$ , the new owner runs the risk of receiving the correct device but being given the wrong secret. In such a case, the device is unusable. The motivation behind this can be for the previous owner to retain control of the device (Threat  $T1$ ) while accessing the new owner's network. Devices should not be connected to sensitive infrastructure before the secret has been verified and changed, and the device has been wiped clean. As these are basic precautions, the attack has really low chances of success. Meanwhile the attacker is no longer in possession of the device. Furthermore, such bad behavior can be reported through the reputation system (Assumption  $A5$ ). There are few incentives to engage in this kind of behavior.

One should be careful about publishing encrypted secrets to the blockchain as attackers may try to decipher them (Threat  $T7$ ). The secret should therefore be updated in accordance with the security of the used encryption scheme. Recovery of old secrets are not a threat as they are only used to activate functions and cease to be useful once they have been updated. This, of course, holds if no information about the new secret can be inferred from the old ones.

## 6.5 Sharing configurations

### 6.5.1 Motivation

We have seen that information pertaining to the device can be stored into the blockchain for its owner's use. It is also the case for information that would be used by users or other IoT devices willing to interact with it. Contrary to static properties such as memory space or power consumption, a device possesses a number of dynamic properties. Such properties can be useful to others for any number of reasons such as authentication, evaluation of trust (and risks), discovery of assets, etc. Other examples include advertising known protocols, the version of the software that is running on a device or other application-specific information.

During the life of a device, the owner can share and update that information using the blockchain.

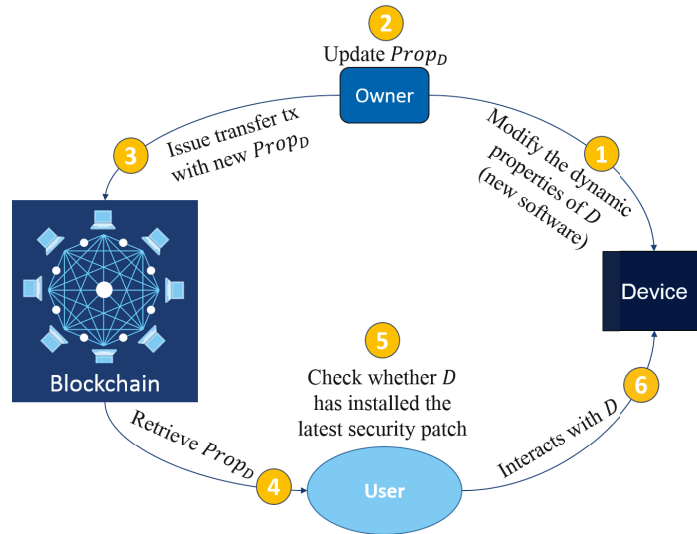


Figure 6.6: Publishing dynamic properties to the blockchain

Transfer transactions can be assorted with any number of properties that the owner judges relevant to the use of their device. Updates are made by transferring ownership to oneself. Users can then retrieve that information from the blockchain.

### 6.5.2 Proposal

Let  $O$  be the owner of a device.  $O$  wishes to publish  $Prop_D$ , a set of dynamic properties of  $D$ . Let  $U$  be a potential user of  $D$ . User  $U$  needs to consult  $Prop_D$  before exchanging with  $D$ . Let  $tx_n$  be the latest blockchain transaction designating  $O$  as  $D$ 's owner. The corresponding output,  $out_l^n$ , must be unspent, otherwise,  $O$  is no longer the owner of  $D$ .

As illustrated in Figure 6.6,  $O$  issues a *transfer* transaction,  $tx_m$ , with  $out_l^n$  as input, signed with  $priv_O$ . This transaction yields one output sent to  $addr_O$  that, additionally, carries  $Prop_D$  (see Table 6.5).  $U$  can then retrieve  $tx_m$  and extract  $Prop_D$ . After verifying that the dynamic properties of  $D$  are in accordance with its requirements,  $U$  interacts with the device.

### 6.5.3 Security analysis and limitations

The first question that comes to mind is the allotted size for  $Prop_D$ . This, once again, depends of the underlying blockchain. In Bitcoin, the maximum size of a block is fixed at 1 MB<sup>43</sup>. Naturally, this limits the size of a transaction. Furthermore, bigger transactions have a higher cost. Miners are paid by transaction fees. If transaction fees were fixed, a large number of small transactions would amount to more fees than a small number of big transactions. To compensate for that, bigger transactions should

<sup>43</sup>Many want to increase this limit but this would require a hard fork. The issue is still being debated.

pay a higher fee. In Ethereum, there is no fixed limit to the size of a transaction but the amount of gas per block is limited. Even if this limit augments with time, the size of a transaction is currently limited to around 100.000 non-zero byte. Bigger  $Prop_D$  also amount to higher cost. Published data should therefore be kept to a minimum. This is inline with many other IoT requirement however.

Privacy concerns might arise from the publication of device information in such public fashion (Threat  $T7$ ). They can be tackled by either using a private blockchain or encrypting the published content to restrict its viewing. Group keys [Harney and Muckenhirm, 1997] or attribute-based encryption [Goyal et al., 2006] can be used to efficiently control access to that information.

Finally, IoT devices may want to retrieve  $Prop_D$ . Unfortunately, the resources required to maintain a connection to the blockchain network are too much for constrained devices at the moment. This limitation is not due to the blockchain technology but rather to its youth. Light clients specifically designed for IoT devices should emerge before long. In the meantime, a gateway can be used to retrieve that information in the device's stead.

## 6.6 Conclusion

The blockchain has made the tracking of asset's ownership relatively inexpensive. It does not have to be reserved for houses and boats any longer. We therefore propose to use it to track the ownership of IoT devices. The chain of ownership can be augmented by adding additional information to transfer transactions. We present two ways to do so. First, encrypted device-related secrets can be added to help the owner manage their devices. Second, relevant dynamic properties can be advertised to users and other devices.

Threats	Attacker	Description
T1	Prev. Owner	Retain access to the device
T2	Prev. Owner	Valid Proof of Ownership but no device is provided
T3	Prev. Owner	Secret is not valid. Provided device cannot be accessed
T4	New Owner	Extract sensitive info from the device
T5	New Owner	Use device to gain access to sensitive data
T6	Third Party	Ownership transfer cannot be completed
T7	Third Party	Access sensitive information by eavesdropping
T8	Third Party	Successfully masquerades as the device owner
T9	Third Party	Create an ownership chain with no corresponding device

Figure 6.7: Threats addressed by our proposal

We have argued the benefits of these applications. Among them, a pseudonymous proof of ownership can be produced to give guarantees to a prospective buyer and replace the registration systems in place, currently run by private companies. Other benefits include the desintermediation and decentralization of classic solutions. Limitations of our proposals have been argued.

A threat model has been detailed and three potential attackers considered: a malicious previous owner, a malicious future owner, and a malicious third party. We have defined six security assumptions and nine security threats involving these attackers. Table 6.7 summarizes the latter. Out of nine threats identified, three are addressed directly by our scheme (in green), three are mitigated (in orange) in particular by the use of a reputation system, and three are not addressed (in red). Threats  $T1$ ,  $T4$ , and  $T5$  however can be mitigated by the precautions taken by device owners right before and after ownership transfers: the erasure of sensitive data, the generation of new secrets, the isolation of a new device before configuration is complete, ...

One could argue that blockchain keys should not be considered safe and can easily be lost or compromised. However, the issue of safekeeping a key has been studied extensively and many solutions can be provided. In the current state of affairs, the requirement that every potential owner possesses a blockchain address seems the most unlikely. We believe this is likely to evolve in a near future.



# Chapter 7

## Conclusion and Perspectives

### Contents

---

<b>7.1 Conclusion</b> . . . . .	<b>185</b>
7.1.1 Research summary . . . . .	185
7.1.2 Research questions . . . . .	189
<b>7.2 Perspectives</b> . . . . .	<b>192</b>
7.2.1 Improvement on our contributions . . . . .	192
7.2.2 Future research directions . . . . .	194

---

This chapter concludes our thesis. Section 7.1 summarizes our research and reflects on the questions stated in Section 1.4. Section 7.2 looks at improvements our contributions could benefit from as well as future research directions.

### 7.1 Conclusion

#### 7.1.1 Research summary

The content of this thesis can be summarized as follows:

**Chapitre 1** introduced our readers to the Internet of Things and its many faces. The IoT is indeed made up of various applications domains such as Smart Gadgets, eHealth, Smart Homes, Smart Buildings, Smart Cities and Infrastructure, or the Industry 4.0. Each area comes with a specific set of requirements, a specific set of users, their own set of legacy systems, values, different properties, etc.

We have also discussed the challenges faced by IoT applications. In an ecosystem where most actors have limited computation power, where bandwidth is restricted either by cost concerns or by the quality of the network itself, and where memory can be a rare commodity, the primary challenge is *resource efficiency*. The *longevity* of systems and application is also a concern: Devices deployed in the wild are hard to maintain while more susceptible to physical attacks. Software updates are difficult to conduct.

Devices should include a potential for growth but resources are already missing for the current task at hand. *Business models* can be hard to find in a fragmented market with emerging regulations concerning technological aspects as well as privacy. But one of the most important challenges facing the IoT today, is its *security*.

Inexperienced users, hard coded passwords, security patches that cannot be delivered, the IoT is becoming infamous for its botch security measures. This not only negatively impacts the growth potential of the field, but also creates numerous unprotected devices that represent free resources for malicious parties. Privacy is a primary concern. But the risks are far greater when it comes to vulnerable medical devices or heavy machinery.

Access control is one of the most important tools in a system's protective arsenal. We defined two central conflicts brimming in the access control community: adaptation vs invention, and centralization vs distribution. IoT systems do not support traditional methods of access control as they are too demanding. One solution is to adapt those methods, which is easier but can lead to cumbersome processes, or invent new ones, harder to do but more likely to fit the different IoT requirements. Centralization is used to circumvent the resource constraints of edge devices but lead to single point of failure and scalability issues. Distribution on the other hand presents a lot of advantages but is hard to achieve when devices are so limited. In some cases however, contacting a central server is not an option: when on a boat in the middle of the ocean for instance. Access control solutions must account for these situations.

The challenges of handling a device's life-cycle have been addressed. from the manufacturing line to its first deployment and eventual retirement, a device may change hands many times. Tracking and facilitating those exchanges is in everybody's best interest. To know where to send security updates for instance.

From these challenges, we have extracted six research questions that the rest of this thesis tried to answer:

1. What are the main remaining challenges of access control in the IoT?
2. What role does architecture play in the existing access control solutions?
3. Can access control logic be deported to edge node to enable serverless authorization decisions?
4. How can we increase security usability?
5. Can the blockchain be leveraged for decentralizing access control while maintaining expressivity and offline access?
6. Using the blockchain, can we tackle other IoT issues such as the lifecycle of devices?

Finally, a summary of our contributions was presented and the organization of the manuscript detailed.

**Chapitre 2** took a deep dive into the blockchain technology. It succinctly introduced the cryptographic primitives behind the blockchain: hash functions, digital signatures, and merkle trees. Beyond basic concepts such as transactions, addresses, miners, or blocks, it presented the diversity and complexity of the blockchain ecosystem.



We discussed the specifics of three of the most popular blockchain implementations, namely Bitcoin, Ethereum, and Hyperledger. Transaction-based model, blockchain accounts, wallets, smart contracts, public and permissioned model, order of execution, etc. Similarities and differences were dissected.

Consensus protocols were heavily featured. We addressed the workings, benefits, and short comings of Proof of Work and Proof of Stake as well as alternative, less common methods. Proof of activity combines the two previous protocols while the Ripple consensus mechanism is based on decentralized trust.

Governance is another touchy issue. First, there is the question of whether to go with an open public blockchain or towards a more restricted permissioned blockchain. If you choose public, then comes the issue of collectively deciding which bugs to fix, what improvements should be made, and how to react to attacks. These decisions can be made off or on the blockchain itself. We also looked at soft and hard forks and their consequences on the future of the community.

Security has been discussed, of course. Famous attacks such as the double spending or 51% attacks. Less famous ones such as withholding attacks. Beyond attacks, potential vulnerabilities and limitations have also been addressed: scalability issues, the legal framework around the blockchain, or privacy concerns.

Armed with this knowledge, the relevance of an association between the blockchain and the IoT has been debated. On the one hand, we have benefits such as decentralization, desintermediation, transparency, reduced cost, and built-in auditability. On the other hand, scalability issues, transactions' validation time, stability, usability, and of course resource efficiency are a concern. But the blockchain is young and some of these issues can be circumvented until the community comes up with a solution.

**Chapitre 3** surveyed access control solutions for the IoT. It presented the background behind access control: the difference between the policy, model, and mechanism abstractions, the four fundamental functions of access control (PEP, PDP, PAP, and PIP), capability-based and attribute-based access control.

We proposed 16 criteria to compare solutions from the literature. Some objective, some qualitative. Our analysis is separated by architecture: centralized, hierarchical, federated, and distributed. These architectures are defined using the multiplicity and localization of the PDP function. For each architecture, we detailed the benefits and shortcomings that solutions inherit from their choice of architecture. Examples from the literature are provided to illustrate our remarks.

Using this analysis, we defined an architecture-based taxonomy of IoT access control solutions to help practitioners pick the architecture that will work the best for their application. Future research directions were also mentioned. Among them, usability, privacy, and serverless authorization, three properties that we tried to integrate in our following contributions.

**Chapitre 4** presented the OATL project, a set of access control libraries to issue, store, and verify authorization tokens. Through a smart car rental use case, we highlighted the need for an access control solution that would offer: resource efficiency, serverless authorization, revocation, granularity, context-awareness, and be actuator-compatible. But our focus was not on the access control process itself. OATL aims at pre-packaging access control mechanisms for IoT developers. From that lense, new requirements emerge: ease of integration, generality (across application domains), and modularity (across many types of devices, protocols, policies, etc).

We proposed three libraries, each hosted on a different actor, that developers would integrate in their solutions. The Token Generation library is hosted on an IoT cloud platform. It issues tokens but does not take authorization decisions. The Client-side library is called by the mobile application. It manages the tokens for the client, including storing sensitive information such as private keys. Finally, the Token Interpretation Information verifies the token's legitimacy and challenges its bearer to prove its ownership of it by requesting a signature over a random message.

We proposed two types of tokens to accommodate the requirements of different applications. Neither carries the identity of its intended recipient. A new token key is generated with each token and acts as short term credentials for the user. The first token uses symmetric cryptography and assumes devices are not time-aware. The second is based on public-key cryptography and assumes time-awareness.

After detailing the access control process and revocation mechanism, we presented a thorough security analysis of our proposal. We highlighted what properties were offered by our libraries and which remained the responsibility of the vendor that would implement the code to run each component. A Proof of Concept has been implemented to showcase the usability of our solution.

We concluded the chapter by a comparison of our solution with the state of the art using the same architecture. OATL distinguishes itself by enabling explicit revocation with serverless authorization, by its focus on usability, and by its respect of privacy.

**Chapitre 5** introduces MAAC-B, another access control system that uses the blockchain instead of tokens. MAAC-B is made of three services: attribute management, policy management, and trust management. Each of them is based on one or several smart contracts that register the corresponding information into the blockchain.

Attribute Contracts are deployed by the clients themselves, giving them the freedom to slip their attributes over however many pseudonyms as they wish. Anybody can endorse an attribute. It is a simple blockchain transaction. The validity of an attribute, its trust level, is then defined by the trust the validating entity puts in the endorsing entities.

The Trust Anchor Service is used to manage trusted entities, and promote new administrators. The Policy Service also uses it to check whether some entity is a registered administrator, and therefore should be able to modify policies. Policies are defined in a generic fashion. They are made of a list of attributes the requestor must possess and the trust level that must be met in order for an attribute to be deemed valid. Dispatch contracts then associated these policies to resources.

Devices are connected to the blockchain via their gateway that scans incoming blocks for policy updates. Policies are then bundled together and transmitted to the device for local storage. When an access request is placed, the device queries its gateway for the requestor's attribute endorsements. Each attribute is then score according to them.

We provide a security analysis of our proposal, and compare it to other blockchain-based IoT access control solutions from the state of the art. Once again, our proposal enables serverless authorization as well as user privacy. Attributes make for a flexible solution with high context-awareness. Their issuance is distributed.

**Chapitre 6** departed from access control and focused instead on ownership. Asset tracking is one of the more natural blockchain applications. Once reserved for expensive items such as jewelry, the blockchain

has democratized the practice to more common assets. IoT devices can benefit from this practice for a few reasons. The desintermediation enabled by the blockchain lets users produce an independent proof of ownership that can be used when selling a device, when making repair, or to install new software. Decentralized storage and processing enhances availability and persistence. Using the transparency and accessibility of ownership transfer information, one could automate permission revocation in the event of an ownership change. Owners control their identity and the information they share about their device.

We proposed two types of transactions: genesis transactions create the link between tangible objects and their digital counterparts, transfer transactions change ownership from the emitter to the recipient. An independent Proof of Ownership can be produced by signing a challenge message with the key that corresponds to the recipient address of the latest transfer transaction.

Two extensions were proposed. The first aimed at improving the management of device-related secrets. We considered the IoT device as a black box providing functions that can only be activated using a secret. This secret must be transferred from the original to the new owner. We proposed to embed it in the transfer transaction, encrypted with a secret key shared by vendor and actor. The secret can then be retrieved from the blockchain and decrypted. The same scheme can be used to manage the secrets of a large number of devices: secrets are encrypted and posted to the blockchain. The owner uses a master key to derive one encryption key for each secret based on the device identifier. Now the owner only needs to safeguard its master key.

The second extension embeds dynamic device properties into transfer transactions. Devices possess a set of static properties such as their overall memory space. They also possess dynamic properties such as the memory space that is currently available. Potential users might be interested in these properties. The blockchain would let owners advertise them without many additional cost.

We defined seven security assumptions and nine potential threats. Each proposal has been analyzed through this spectrum. We also addressed limitations.

### 7.1.2 Research questions

In Section 1.4, we asked six research questions and have tried to answer them throughout this manuscript. Table 7.1 summarizes which contributions tackles what research question. Here is what we found.

Questions	Contributions			
	1	2	3	4
1	x			
2	x			
3	x	x	x	
4		x		x
5	x		x	
6				x

Table 7.1: Research questions in contributions

**Question 1** *What are the main remaining challenges of access control in the IoT?*

In Chapter 3, we analyzed the state of the art in IoT access control. We concluded that the community was focused on either fully centralized or distributed solutions, leaving a lot of space for hybrid architectures. Federation seems to be getting more interest recently. If the world is heading towards smart infrastructure, *interoperability* needs to be addressed.

*Usability* is another important property that the community should focus on. Without it, security measures are simply disabled by end-users. When dealing with companies, we can rely on their policy to push employees to safeguard company's interests and apply security measures. When end-users are individuals, only usability can ensure compliance. Yet usability is still not integrated in the design of proposals. It is not the case for every solution though. A number of solutions try for instance to reduce the number of time a user has to authenticate by isolating the authentication process from the authorization process. Default password should be stronger and unique, written on the device's packaging, as is the case for Internet routers. Other solutions use context information for continuous, background access control. User involvement should be minimized.

Users also demand *privacy*. The European General Data Protection Regulation (GDPR<sup>44</sup>) requires opt-out options when it comes to user information collection. This is not feasible in many existing solutions, as the systems are not built to enable that. Because privacy was not considered as part of the system's requirements. In the future, privacy should be integrated at the design phase.

In Section 1.2, we list some of the challenges the IoT is facing. One of them is longevity. This translates to the access control process as *flexibility*. Because of the constrained nature of IoT devices, access control proposals tend to be hyper specific. They tend to lack adaptability and do not account for potential changes in organization or requirements.

Finally, *serverless authorization* needs to be addressed in more details. In the state of the art, solutions that offer serverless authorization lack other properties that are present in solutions that do not support serverless authorization. Explicit revocation is an example. But there are many use cases that would benefit from serverless authorization and cannot afford the current drawbacks.

## **Question 2** *What role does architecture play in the existing access control solutions?*

This question is answered via our exploration of the state of the art in Chapter 3. We have shown that a lot of a solution's properties are inherited from their architecture. Beyond the obvious aspects that are scalability or resilience, it plays a role on how revocation is addressed, how delegation can be achieved, or what level of context-awareness can be attained.

In Section 3.4, we put forward the properties shared by centralized architectures, differentiating between those who enable serverless authorization (Section 3.4.2), and those who don't (Section 3.4.1). Section 3.5 defines hierarchical architectures and provides the benefits and shortcomings of such architectures for the IoT. Federation is addressed in Section 3.6. Distributed architectures (Section 3.7) proved harder to define. They have therefore been split into three types, each with their own sets of properties: PDP/PEP hybrid (Section 3.7.1), multi PDP (Section 3.7.2), and blockchain-based solutions (Section 3.7.3)

Our survey concludes that architecture is a good indicator of similarities between IoT access control solutions.

---

<sup>44</sup>GDPR: <https://gdpr-info.eu>, Last Checked: July 28th, 2019

**Question 3** *Can access control logic be deported to edge node to enable serverless authorization decisions?*

This thesis answered that question in three ways. First, it explored the state of the art for solutions enabling serverless authorization. Then, we proposed two solutions of our own: OATL and MAAC-B.

The survey presented in Chapter 3 has shown there are several ways to achieve serverless authorization with various degrees of edge intelligence. Centralized solutions (see Section 3.4.2) use tokens to transfer the access control decision from the PDP to the PEP, located in the device, thus enabling serverless authorization. Here, the edge device only enforces the decision. In Section 3.7.1, we have seen PDP/PEP hybridization, allowing the device to take part in the access control decision by checking extra conditions at access time.

OATL takes the hybridization approach to edge intelligence. In chapter 4, we propose a set of libraries to manage tokens. Issuers have the option to embed local conditions into the tokens. OATL however does not provide a format for these conditions, nor is it part of our libraries' duty to handle the evaluation.

MAAC-B on the other hand, presents a solution where the device acts as an independent PDP. Chapter 5 defines a policy management system that lets administrators push policies onto their devices, acting as PAP. The blockchain, by storing user's attributes, acts as a PIP for the device. It is the device that ultimately evaluates each attribute's trust level and evaluates the policy, pushing the access control logic to the device.

**Question 4** *How can we increase security usability?*

Usability is mainly addressed by the OATL project presented in Chapter 4. Usability is generally considered with regards to the user. But the security issue in the IoT is partially due to the absence of adequate security measures within IoT products. We therefore considered the problem of usability at the design phase. The goal then becomes to push IoT vendors to integrate robust access control mechanisms in their product. To achieve this, we focused on a solution that would be inexpensive and easy to integrate. We built a solution that is as generic as possible and can be used outside of the car rental use case we used as illustration.

The authorization engine and ultimate access decision are purposefully left out of our scope to accommodate different requirements. Several types of tokens are proposed to accommodate different types of devices. A library is proposed for each actor in the process. The model is therefore valid whether a single actor is in charge of developing all three pieces of software (cloud platform, mobile application, and device application) or each brick is developed independently. Interoperability is assured by the use of the OATL libraries.

Chapter 6 also touches on usability. It provides a simple method to track ownership and handle a device's lifecycle at a reduced cost. The blockchain provides a shared infrastructure that frees users from vendor dependency. Section 6.4 adds on the original proposal by contributing a method to manage device-related secrets.

**Question 5** *Can the blockchain be leveraged for decentralizing access control while maintaining expressivity and offline access?*

Blockchain-based solutions to IoT access control were surveyed in Section 3.7.3. We found that context-awareness and expressivity were highly dependent on the specifics of each solution. Two proposals [Ouaddah et al., 2016b, Zhang et al., 2018b] offer offline access. Ouaddah et al. require the involvement of the device owner and the issuance of several blockchain transactions for each access which can be a deal breaker for time sensitive solutions. Zhang et al. use ACL to take access control decisions. Their ACL are static and hyper specific, leading to a decrease in expressivity.

Chapter 5 describes our own attempt at using the blockchain as a tool for decentralization. MAAC-B uses the ABAC model. Attributes in ABAC can be used to represent any and all features of subjects, objects, actions, or of the environment: identity, status, capacities, location, role, etc. This makes ABAC a highly expressive system. If MAAC-B is only focused on user attribute, it still allows more flexibility than static ACL. It requires a working connection to the device's gateway which is a requirement found in many IoT applications. The system can operate normally when cut off from the rest of the blockchain. Updates are not received of course. But offline access is successfully enabled.

**Question 6** *Using the blockchain, can we tackle other IoT issues such as the lifecycle of devices?*

This question is addressed by Chapter 6. In this chapter, we use the blockchain to track ownership changes, obtain device-related secrets when they arise, and advertise changes in dynamic properties of devices. Compared to current ownership registration systems, the blockchain offers desintermediation, decentralization, transparency of processes, and an increase in both usability and privacy.

## 7.2 Perspectives

### 7.2.1 Improvement on our contributions

In this section, we address potential extensions for our work.

**Contribution 1: Access control survey** By nature, a survey can always be extended by adding newer proposals. IoT access control is a hot topic with new solutions coming out every month. Blockchain-based solutions are especially popular considering that there were none only five years ago.

Outside from the obvious then, we would like to refine our survey by providing a detailed decision tree that would not only guide the reader to the type of architecture that best fits their needs, but also to the proposal that best matches their requirements.

**Contribution 2: OATL** The OATL project is very much ongoing and the results presented in Chapter 4 only represent the first iteration. As stated in the conclusion of this chapter, improvements to OATL can be classified in two categories: improvements to the existing implementation, and improvements to the proposal.

To improve our implementation, the following should be added to the PoC:

- **Update type #1 tokens** - Section 4.3.1 presents a specification that has been improved by feedback from the PoC implementation. Parts of the revised specification are:

- New derivation algorithm
- Challenge keys
- Clearer derivation process for session keys
- **Implement type #2 tokens** - The PoC only provides type #1 tokens. Implementing type #2 tokens is important to showcase the modularity of our solution.
- **Focus on resource efficiency** - The first iteration was focused on usability. Resource efficiency should be the focus of the second iteration as it is paramount for IoT applications.
- **Test local conditions** - The influence of local condition evaluation on resource efficiency should be evaluated.
- **Change test conditions** - Our current PoC does not involve realistic devices. Because of the emphasize we put on usability, actual IoT devices should be used to test our solution.

Additionally, our proposal can be improved by addressing the following topics:

- **Binary format** - Our tokens use the JSON format. JSON can be verbose. More restricted applications might prefer a binary format for the tokens to reduce message overhead. Such a format would be less flexible and harder to interpret but more efficient.
- **More token types** - One of the aspects of our proposal is modularity. To deliver on this premise, more token types should be added to the proposal. These types might include tokens that carry subject's identity, bearer tokens, etc.
- **App-less access** - Serverless authorization is an important part of our proposal. However, it is harder and harder to get users to install dedicated applications on their mobile phone. For more usability, OATL would benefit from an app-less option. The mobile phone would then acts as a relay between the device and the cloud platform. This would disable serverless authorization but increase usability.
- **Explicit revocation for type #2 tokens** - Explicit revocation is only enabled with type #1 tokens. We would like to extend this feature to type #2 tokens. This can be achieved by embedding revocation inside any and all new authorization tokens. The revocation would then be delivered to the device when any new token is used by anyone after the revocation has been issued. This requires a compact representation of token revocation.
- **Delegation** - The delegation of permissions between users would alleviate some of the administrative work, enhance usability, and privacy: a hotel guest could invite a guest overnight, duplicate its virtual key for the occasion, and revoke it in the morning, all without disclosing the guest to the hotel staff.
- **Access logs** - Devices are constrained memory-wise and cannot be supposed to connect to the cloud platform after the initial bootstrap phase. Without a gateway, it is interesting to explore how and where to store access logs for the device.

**Contribution 3: MAAC-B** There are four ways in which MAAC-B can be improved:

- **Implementation** - We do not yet provide an implementation for MAAC-B. This is first on the list of future evolution for the solution. An implementation would validate the feasibility of our solution.
- **Accountability system** - Our proposal references a reputation system. This system would benefit from records of past behavior, good and bad. Access control could then be contingent on proof of good behavior.
- **More context** - In its current state, our proposal only focuses on subject's attributes. Our scheme can be extended to include all kind of attributes such as environmental or object-related attributes. This would increase context-awareness.
- **Implicit revocation** - To mitigate the delay in revocation processing, attribute endorsement transaction might be enriched with an expiration time, thus allowing an attribute to be revoked without the need for a new blockchain transaction.

**Contribution 4: Ownership tracking** An interesting extension proposed for our ownership tracking system is the automatic revocation of allotted permissions triggered on ownership change. This system would need to be configured with a list of potential pseudonyms to avoid a false trigger on a transfer transaction meant to either update device-related secrets or hide the owner's identity by switching pseudonyms.

### 7.2.2 Future research directions

Beyond our contributions, we list potential directions for future research.

**A resource-efficient blockchain** At the moment, IoT devices cannot connect directly to the blockchain for lack of resources. There are some attempts at creating IoT-compatible blockchains [Popov, 2016]. This is a really interesting field that is opening up.

**Privacy concern** Privacy is one of the next challenge not only in IoT but in computer science in general. Privacy by design is becoming a requirement. There are privacy concerns surrounding the blockchain. Addressing them outside of permissioned model will be an interesting challenge.

**Delegation** Delegation is rarely mentioned in IoT access control solutions, apart from solutions that solely focus on delegation itself. It has the power to improve usability, increase scalability and manageability, as well as boost privacy. It is therefore worth studying in more details.



**Explicit revocation with serverless authorization** We have seen that explicit revocation is hard to achieve with serverless authorization as the server naturally tends to handle revocation. Explicit revocation is a feature that any use case should enjoy. This is therefore a challenge that should be addressed. We have mentioned the possibility of using clients to carry revocation from the server to the device. It is a trail that is well worth exploring.



# Appendices



# Appendix A

## Publication List

### A.1 Publications in journals, Conferences and Workshops

- **International conference papers**

[Dramé-Maigné et al., 2018] Dramé-Maigné, S., Laurent, M., Castillo, L., and Ganem, H. (2018). Augmented chain of ownership: Configuring IoT devices with the help of the blockchain. In *International Conference on Security and Privacy in Communication Systems*, pages 53–68. Springer.

[Dramé-Maigné et al., 2019] Dramé-Maigné, S., Laurent, M., and Castillo, L. (2019). Distributed Access Control Solution for the IoT based on Multi-endorsed Attributes and Smart Contracts. In *2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC)*, pages 1582–1587. IEEE.

### A.2 Work under review

- **International journal papers**

[Dramé-Maigné et al., 2019+] Dramé-Maigné, S., Laurent, M., Castillo, L., and Ganem, H. (2019+). Centralized, Distributed and everything in-between: Reviewing access control solutions for the IoT. Revised to *Computing Surveys (CSUR)* due to Major revision decision. ACM.



## Appendix B

# Table of acronyms

<b>Acronym</b>	<b>Expanded</b>
6LoWPAN	IPv6 Low power Wireless Personal Area Network
ABAC	Attribute-Based Access Control
AC	Access Control
ACE	Access Control Entry
ACL	Access Control List
AES	Advanced Encryption Standard
AIE	Attribute Issuing Entity
API	Application Programming Interface
a.k.a	also known as
AROP	Authorization Route Optimization Problem
ARPP	Authorization Route Planning Problem
ASIC	Application-Specific Integrated Circuits
ASM	Ambient Space Manager
ASP	Authorization Service Provider
authN	authentication
authZ	authorization
AttC	Attribute Contract
AWS	Amazon Web Service
BAN Logic	Burrows-Abadi-Needham Logic
BIP	Blockchain Improvement Proposal
BTC	Bitcoin
BTLE	Bluetooth Low Energy
CapBAC	Capability-Based Access Control
CCAAC	Capability-based Context Aware Access Control
CN	Coordination Node
CoAP	Constrained Application Protocol
COCapBAC	Community-driven Capability-Based Access Control

CP-ABE	Cyphertext Policy Attribute-Based Encryption
CPU	Central Processing Unit
CSL	Client-Side Library
CSP	Cloud Service Provider
DAC	Discretionary Access Control
DAG	Directed Acyclic Graph
DAO	Decentralized Autonomous Organization
DCAPBAC	Distributed Capability-Based Access Control
DDoS	Distributed Denial of Service
DisC	Dispatch Contract
DNS	Domain Name System
DoS	Denial of Service
DTLS	Datagram Transport Layer Security
EAA	Entité Apprôbatrices d'Attributs
ECC	Elliptic Curve Cryptography
ECDSA	Elliptic Curve Digital Signature Algorithm
EIP	Ethereum Improvement Proposal
ETC	Ethereum Classic
ETH	ETHer
EU	European Union
EVM	Ethereum Virtual Machine
GDPR	General Data Protection Regulation
GPS	Global Positioning System
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HMAC	Hash-based Message Authentication Code
HSM	Hardware Security Module
IBE	Identity-Based Encryption
IdO	Internet des Objets
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
IPoO	Independent Proof of Ownership
IWCMC	International Wireless Communications & Mobile Computing
JSON	JavaScript Object Notation
JWE	JSON Web Encryption
JWS	JSON Web Signature
JWT	JSON Web Token
LoRaWAN	Long Range Wide-Area Network
MAAC-B	Multi-endorsed Attribute Access Control using the Blockchain
MAC	Mandatory Access Control
MAC address	Media Access Control address



---

MD5	Message Digest 5
MitM	Man-in-the-Middle
NGAC	Next Generation Access Control
NIST	National Institute of Standards & Technology
NFC	Near Field Communication
OATL	Offline Authorization Token Libraries
OM-AM	Object Model-Architecture Mechanism
PAP	Policy Administration Point
PBKDF2	Password-Based Key Derivation Function 2
PDP	Policy Decision Point
PdPI	Preuve de Propriété Indépendente
PEP	Policy Enforcement Point
PIP	Policy Information Point
PKG	Private Key Generator
PKI	Public Key Infrastructure
PoA	Proof of Activity
PoC	Proof of Concept
PolC	Policy Contract
PoO	Proof of Ownership
PoS	Proof of Stake
PoW	Proof of Work
PUF	Physical Uncloneable Function
PSK	Pre-Shared Key
QR Code	Quick Response Code
RBAC	Role-Based Access Control
REST	REpresentational State Transfer
RFID	Radio Frequency IDentification
SEAC	Sensing Enabled Access Control
SAML	Security Assertion Markup Language
SIM	Subscriber Identity Module
SQL	Structured Query Language
TBAC	Trust-Based Access Control
TBD	To Be Determined
TCP	Transmission Control Protocol
TE	Trusted Entity
TGL	Token Generation Library
TIL	Token Interpretation Library
TL	Trust Level
TLS	Transport Layer Security
TrAnC	Trust Anchor Contract
tx	transaction
U2IoT	Unit and Ubiquitous IoT

*Appendix B. Table of acronyms*

---

UCON	Usage Control
UMA	User Managed Access
UNL	Unique Node List
URI	Uniform Resource Identifier
UTXO	Unspent Transaction Output
VIP	Very Important Person
WoT	Web of Things
XACML	EXtensible Access Control Markup Language

---

Table B.1: Acronyms

## Appendix C

# Blockchain et contrôle d'accès : Vers un Internet des Objets plus sécurisé - Résumé

### Contents

---

C.1 Introduction . . . . .	205
C.2 Survol du contrôle d'accès dans l'IdO . . . . .	207
C.3 Bibliothèques de jetons d'autorisation hors-ligne . . . . .	207
C.4 Contrôle d'accès à base d'attributs à approbations multiples utilisant la blockchain	209
C.5 Le cycle de vie des objets de l'IdO : Changement de propriétaire et configuration à distance . . . . .	210
C.6 Conclusion . . . . .	211

---

### C.1 Introduction

L'Internet des objets désigne l'interface entre le physique et le digital. Dans ce contexte, un *objet* est un objet physique équipé de capteurs, de logiciels, et autres équipements électroniques. Ces équipements lui permettent d'échanger des données avec d'autres objets, des serveurs situés dans le cloud, ou de les présenter directement aux utilisateurs. Dans certains cas, les objets connectés sont aussi utilisés pour agir sur leur environnement.

Avec plus de 21.5 milliards d'objets estimés d'ici 2025<sup>45</sup>, l'IdO est en pleine expansion. Ce nombre n'inclut pas les smartphones, les tablettes, les ordinateurs portables, et autres terminaux utilisés pour interagir avec les objets en question.

L'IdO a de nombreux domaines d'applications tels que la santé, la domotique, ou encore l'infrastructure des villes de demain. Naturellement, ce domaine s'adresse donc à des clients multiples. Cette hétérogénéité de cas d'usage et d'utilisateurs constitue un défi dans le déploiement de solutions pour l'IdO, de même que les faibles capacités de calcul ou de mémoire dont disposent les objets.

---

<sup>45</sup>Estimation par IoT Analytics, voir Figure 1.1

Mais le plus grand défi de l'IdO reste sa sécurité. Ce manque de sécurité menace la vie privée des utilisateurs, parfois leur sécurité physique dans le cas d'objets médicaux ou d'applications industrielles, et fournit des ressources potentielles à des acteurs malveillants. Cette thèse se concentre sur les problématiques liées au contrôle d'accès et au cycle de vie des objets.

Cette thèse propose de répondre aux questions suivantes:

1. Quels sont les principaux défis du contrôle d'accès dans l'IdO ?
2. Quel rôle joue l'architecture dans les solutions de contrôle d'accès existantes ?
3. La logique de contrôle d'accès peut-elle être déportée vers le bord du réseau pour se passer du serveur lors de la demande d'accès ?
4. Comment peut-on améliorer l'ergonomie des solutions de sécurité ?
5. Peut-on utiliser la blockchain pour décentraliser le contrôle d'accès tout en conservant l'expressivité du système et un accès hors-ligne ?
6. L'usage de la blockchain, permet-il d'aborder d'autres problèmes de l'IdO tel que le cycle de vie des objets ?

Pour répondre à ces questions, nous proposons quatre contributions.

La première est une étude approfondie des solutions actuelles de contrôle d'accès de l'IdO. Notre analyse s'articule autour de l'architecture choisit par les auteurs. Une taxonomie est proposée ainsi que des directions pour de future recherches sur le sujets. Cette contribution répond aux questions 1, 2, 3, et 5 définies ci-dessus. Elle est résumée dans la Section [C.2](#).

Notre seconde contribution propose un ensemble de trois bibliothèques utilisées pour générer, stocker, et vérifier des jetons d'autorisation. Ces bibliothèques ont vocation à aider les développeurs de produits IdO à intégrer des mécanismes de contrôle d'accès dans leur solution. Ce faisant, nous répondons aux questions 3 et 4. Cette contribution est détaillée dans la Section [C.3](#)

Notre troisième contribution présente une seconde solution de contrôle d'accès. Nous utilisons la blockchain pour gérer les attributs des utilisateurs, les politiques de contrôle d'accès ainsi que la confiance entre les différentes entités. Cette contribution répond aux questions 3 et 5. Elle est présenté dans la Section [C.4](#).

Notre dernière contribution s'intéresse aux propriétaires d'objets connectés. Nous définissons une preuve de propriété indépendante basé sur des transactions blockchain. Ce faisant, l'archivage des changements de propriétaires est décentralisé et peut s'effectuer sans intermédiaire. Le même système peut être utilisé pour transmettre et gérer les secrets liés aux objets (un code PIN par exemple), ou pour diffuser les propriétés dynamiques d'un objet qui pourraient intéresser des utilisateurs potentiels (i.e. protocoles de communication, version de l'OS installée, etc). Ceci réponds aux questions 4 et 6. Ces travaux sont résumés en Section [C.5](#).

## C.2 Survol du contrôle d'accès dans l'IdO

La plupart des études sur la sécurité de l'IdO mentionnent le contrôle d'accès plus ou moins superficiellement. Ouaddah et al. [Ouaddah et al., 2017b] ont été les premiers à en faire le sujet d'un article complet. Nous prenons ici une approche différente de la leur, basée sur l'architecture, similaire à celle employée par Roman et al. [Roman et al., 2013] dans un article traitant de sécurité au sens large.

Les mécanismes de contrôle d'accès peut être découpé en quatre fonctions fondamentales :

- le **PAP (Policy Administration Point)** crée les politiques d'accès,
- le **PDP (Policy Decision Point)** prends les décisions,
- le **PEP (Policy Enforcement Point)** mets en oeuvres les décisions prises par le PDP,
- le **PIP (Policy Information Point)** est parfois consulté lors de la prise de décision pour renseigné le PDP sur les attributs du sujet, de l'objet ou de l'environnement. Plusieurs PIP peuvent être consulté.

La répartition de ces fonctions entre les différents acteurs ainsi que leur multiplicités définissent nos différentes architectures. Nos définitions se concentrent sur le PDP particulièrement. C'est en effet la fonction qui demande le plus de ressources : de la mémoire pour stocker les politiques de contrôle d'accès, de la puissance de calcul pour faire tourner le moteur de décision, de la bande passante pour interroger les PIP ou communiquer la décision au PEP.

Nous définissons sept architectures réparties en quatre catégories : centralisée, hiérarchique, fédérée, et distribuée. Les solutions adoptants une architecture centralisée sont particulièrement sensibles à la santé du serveur central abritant le PDP. Ainsi, nous distinguons les solutions qui nécessitent son intervention lors du contrôle d'accès (solutions en ligne) de celles qui peuvent s'en passer (solutions hors ligne). Les solutions distribuées sont elles aussi divisées en plusieurs catégories. Lorsque l'objet conserve une partie de son autonomie et est en charge d'une partie de la décision de contrôle d'accès, nous parlons d'hybridation PDP/PEP. Lorsque plusieurs PDP sont accessibles, nous parlons de solutions multi-PDP. Enfin, nous distinguons les solutions qui utilisent la blockchain pour distribuer les fonctionnalités du PDP.

Les solutions de l'état de l'art sont analysées et comparées à l'aide de seize critères différents. Certains sont qualitatifs, d'autres quantitatifs. Notre analyse donne lieu à une taxonomie basée sur l'architecture. Nous identifions également des directions propices à de futures recherche. Parmi elles, l'ergonomie des solutions, la vie privée des utilisateurs, et la possibilité d'effectuer le contrôle d'accès en l'absence du serveur lorsqu'il existe.

## C.3 Bibliothèques de jetons d'autorisation hors-ligne

En accord avec les conclusions tirées de l'état de l'art, notre première solution de contrôle d'accès se concentre sur l'ergonomie. Nous ciblons ici les développeurs qui sont les premiers responsables du manque de sécurité dans les produits de l'IdO. Nous leur proposons un ensemble de bibliothèques faciles à intégrer qui se chargent des mécaniques de contrôle d'accès. Le code applicatif n'a alors plus qu'à

faire appel à ces bibliothèques pour profiter de leurs garanties de sécurité. Notre solution se veut près des usages et des architectures existantes.

Considérons le cas d'une société de location de voitures connectées. Les clients peuvent réserver leur voiture via une application installée sur leur téléphone portable, en choisissant notamment leurs options (GPS, Air Conditionné, etc). A l'issue de la réservation, l'application communique avec un serveur cloud pour obtenir une clé virtuelle. Pas besoin d'aller chercher la voiture dans une antenne de la société, l'application indique la voiture la plus proche correspondant aux critères du clients. Lors de l'utilisation, les fonctionnalités de la voiture sont débloquentées en fonction des droits inclut dans la clé virtuelle. Cette clé doit fonctionner même en l'absence de couverture réseau, donc sans accès au serveur cloud. A la fin de la période de location, le client utilise de nouveau l'application qui extrait les données d'utilisation de la voiture et facture l'utilisateur au plus juste. Après ça, la clé virtuelle doit cesser de fonctionner.

Pour répondre à ce cas d'usage, notre solution doit posséder les propriétés suivantes : efficacité dans l'utilisation des ressources, contrôle d'accès hors ligne, possibilité de révoquer les droits d'accès, connaissance du contexte, et granularité dans les autorisations. De plus, notre solution doit fonctionner avec les objets qui agissent sur leur environnement, en opposition aux capteurs qui se contentent de produire des données qui peuvent ensuite être traitées ailleurs.

Dans un soucis d'ergonomie à l'usage des développeurs de produits connectés, notre solution doit posséder quelques propriétés supplémentaires : facilité d'intégration, compatibilité avec plusieurs domaines d'application, et compatibilité avec différents types d'objets, protocoles, modèle de contrôle d'accès, etc.

Nous utilisons des jetons pour matérialiser l'autorisation. Cette solution est à la fois très utilisée dans l'IdO et compatible avec notre modèle : l'émission et la gestion des jetons peuvent être confiées à nos bibliothèques tandis que leur transmission restent à la charge du code applicatif.

Le contrôle d'accès peut être séparé en deux segments : l'obtention du jeton et la demande d'accès proprement dite. Le premier segment n'implique que le mobile et le serveur cloud, le second n'implique que le mobile et l'objet connecté.

Nos bibliothèques sont aux nombres de trois, une pour chaque acteur. La première se charge de l'émission des jetons et est utilisée par le serveur cloud. Elle n'agit pas en tant que PDP. Ainsi, le choix du moteur de décision reste libre pour le serveur cloud. Une fois la décision d'accès prise, notre bibliothèque la matérialise sous forme de jeton.

La seconde bibliothèque est utilisée par le code de l'application mobile. Elle se charge de stocker les jetons et les informations privées associées de manière sécurisé. Cette bibliothèque ne s'occupe pas de formuler les requêtes. Elle est par contre appelée pour en préparer le contenu. Ceci permet de ne pas contraindre les choix des protocoles de communication.

La dernière bibliothèque vérifie la validité des jetons, c'est à dire à la fois leur intégrité et la légitimité de l'utilisateur à les utiliser. Elle est située sur l'objet connecté. Cette bibliothèque n'interprète pas les droits contenus dans le jeton. Notamment, elle ne vérifie pas l'adéquation entre le jeton et la requête. En outre, le jeton peut contenir des conditions à vérifier localement. Cette évaluation reste à la charge du code applicatif, pour plus de liberté dans l'expression des conditions.

Nous proposons deux types de jetons pour s'adapter à des objets plus ou moins contraints. Les jetons ne comportent pas de référence explicite à l'identité de leur propriétaire. De nouvelles clés cryptographiques sont générées pour chaque nouveau jeton. Le premier utilise de la cryptographie symétrique

et considère que l'objet n'a pas de notion du temps. Le second type de jeton utilise de la cryptographie asymétrique et inclut une date d'expiration.

Notre solution comprend une analyse de sécurité détaillés ainsi qu'une Preuve de Concept (PdC) illustrant la facilité d'intégration de la solution.

## **C.4 Contrôle d'accès à base d'attributs à approbations multiples utilisant la blockchain**

Notre troisième contribution propose une seconde solution de contrôle d'accès qui adopte cette fois une architecture distribuée. La blockchain est un registre distribué qui n'accepte que les ajouts. Ce registre enregistre et ordonne des transactions soumises par ses utilisateurs. La blockchain est le nom donné à la fois au registre et au réseau d'utilisateur qui le constitue. Nous utilisons ici la blockchain pour gérer trois services: les attributs des utilisateurs, les politiques de contrôle d'accès, et la confiance entre les différents acteurs.

Les différents services utilisent des contrats intelligents. Ce sont des protocoles informatiques qui facilitent et assurent l'exécution des termes d'un accord entre différentes entités. Dans la blockchain, les contrats intelligents sont exécutés par tous les noeuds du réseau. Leur code est accessible à tous et ne peut pas être modifié une fois déployé.

Les utilisateurs déploient eux-même les contrats utilisés pour contenir leurs attributs. Ce faisant, ils conservent la gestion de leur identité numérique. En effet, un contrat s'assimile à un pseudonyme. L'utilisateur peut alors choisir quelle identité donner à chaque entité avec laquelle il interagit. Ces contrats sont utilisés par les Entité Approbatrices d'Attributs (EAA). N'importe qui peut devenir une EAA en générant une transaction approuvant un attribut pour un client donné.

Deux types de contrat sont utilisés pour gérer les politiques de contrôle d'accès. Le premier associe attributs et niveaux de confiance pour créer des règles d'accès: L'accès est autorisé si l'utilisateur peut justifier de la possession de tous les attributs avec le niveau de confiance associé à chacun. Le niveau de confiance est utilisé pour paramétrer l'accès. Il est basé sur la niveau de confiance accordé aux EAA ayant approuvés un attribut par l'objet qui l'évalue. Le deuxième contrats associe chaque ressource à une politique de contrôle d'accès stockée dans un contrat du premier type. Ces contrats sont déployés et utilisés par des administrateurs.

Enfin, la confiance entre les différentes entités est gérer par un seul type de contrat. Ces contrats sont déployés par un administrateur. Ils peuvent ensuite être utilisé pour promouvoir un nouvel administrateur ou encore accorder la pleine confiance à certaines EAA. Les contrats définissant les politiques de contrôle d'accès invoquent ces contrats pour s'assurer qu'un utilisateur est bien un administrateur. Un service de réputation détermine le niveau de confiance accorder aux EAA auxquelles les administrateurs ne font pas totalement confiance.

Les objets sont connectés à la blockchain par l'intermédiaire d'une passerelle qui prend part au réseau. Cette passerelle filtre les nouvelles transactions et compile les mises à jour concernant les objets qui lui sont associés. Ces mises à jour sont envoyées périodiquement ou récupérées sur demande.

Lors de la demande d'accès, le client prends l'identité de son choix et joint l'adresse du contrat correspondant dans la blockchain à sa demande. L'objet garde localement la version des règles d'accès

correspondant à la dernière mise à jour. Il identifie les attributs nécessaires et contacte la passerelle pour récupérer les approbations des EAA stockés dans la blockchain. Pour chaque attributs, les approbations sont vérifiées et un niveau de confiance est déterminé en fonction de la confiance accordées aux EAA approbatrices. Si le niveau de confiance est supérieur ou égal à celui demandé par les règles d'accès, l'attribut est validé. Si le client possède tous les attributs nécessaire, la demande d'accès est acceptée.

Notre contribution comprends une analyse de sécurité détaillée de notre solution qui évalue notamment sa sensibilité aux menaces telles que le vol d'identité, les approbations frauduleuses, la suppression ou l'élévation de privilège, ou encore les attaques par jeu.

## **C.5 Le cycle de vie des objets de l'IdO : Changement de propriétaire et configuration à distance**

Le suivi des actifs est l'une des premières applications de la blockchain. Cette pratique est en générale réservée aux objets précieux tels que les oeuvres d'art, les biens immobiliers, ou encore les bijoux. La blockchain la démocratise et la rend applicable à des objets de moindre valeur tels que les objets connectés.

Son application à l'IdO présente plusieurs avantages. La blockchain permet de se passer d'intermédiaire, ce qui réduit les coûts et ouvre la porte à une Preuve de Propriété Indépendante (PdPI). Cette preuve se base sur des transactions enregistrées dans la blockchain et ne nécessite pas de connaître l'identité de l'utilisateur, seulement son pseudonyme. Une PdPI peut être demandée lors de la revente d'un objet connecté, pour effectuer des réparations ou modifications importantes, ou encore pour installer un nouveau logiciel.

Le stockage et le traitement décentralisé du registre renforce sa disponibilité ainsi que sa persistance. Les modifications illicites sont plus difficiles à opérer. La transparence et accessibilité des information de changement de propriétaire peut être utilisé pour automatiser les révocations de droits associer à un objet. Enfin, les propriétaires contrôlent eux même leurs identités ainsi que les informations qu'ils partagent sur leurs objets.

Nous proposons deux types de transactions : les transactions de genèse créent le lien entre l'objet tangible et son équivalent digital, les transactions de transfert changent la propriété d'un objet de l'émetteur vers le récepteur. Une PdPI peut être produite en signant un message généré aléatoirement avec la clé correspondant au récepteur de la dernière transaction de transfert lié à un objet.

Nous proposons deux extensions. La première vise à améliorer la gestion des secrets liés aux objets. Nous considérons l'objet comme une boîte noire dont les fonctions ne peuvent être activées qu'à l'aide d'un secret. Ce secret doit être transféré du propriétaire original vers le nouvel acquéreur. Nous proposons d'intégrer ce secret dans la transaction de transfert, chiffré à l'aide d'une clé partagée par le vendeur et l'acheteur. Le secret peut ensuite être récupéré depuis la blockchain et déchiffré.

Le même mécanisme peut être utilisé pour gérer les secrets d'un grand nombre d'objets : les secrets sont chiffrés et postés dans la blockchain. Le propriétaire utilise une clé maître pour dériver les clé de chiffrement de chaque secret à l'aide de l'identifiant de l'objet. Cette clé est alors la seule que le propriétaire stocke et protège.

La seconde extension intègre les propriétés dynamiques de l'objet à la transaction de transfert.



Chaque objet possède des propriétés statiques telles que leur espace mémoire ou leur processeur. Un objet possède aussi des propriétés dynamiques que sont l'espace mémoire disponible ou encore les versions des logiciels actuellement installés. Des utilisateurs potentiels pourraient être intéressés par ces propriétés. La blockchain permet aux propriétaires de les recenser et de les diffuser sans sur-coût.

Notre contribution contient une analyse de sécurité et discute les limites de nos propositions.

## C.6 Conclusion

En conclusion, nous proposons des améliorations à chacune de nos contributions ainsi que des directions pour continuer nos travaux de recherche.

Notre étude de l'état de l'art peut naturellement être mise à jour avec des articles plus récents. De nouvelles solutions de contrôle d'accès paraissent régulièrement, notamment des solutions de fédération ou utilisant la blockchain. Une autre possibilité serait d'affiner notre taxonomie en proposant un arbre de décision dont les branches seraient non plus des architectures mais des solutions.

Concernant nos bibliothèques à l'usage des développeurs, deux pistes peuvent être suivies : améliorer l'implémentation ou améliorer la proposition en elle-même. En effet, notre implémentation n'inclut actuellement que le premier type de jeton. De plus, nos tests n'incluent pas, par exemple, l'évaluation des conditions locales. La proposition quant à elle pourrait être enrichie par différents format de jetons pour s'adapter aux objets les plus contraints, ou encore par l'ajout de mécanismes de délégation des autorisations.

Notre solution à base d'attributs gérés par l'intermédiaire de la blockchain gagnerait à être implémenté. De plus, la proposition pourrait être enrichie par un service enregistrant les actions passés des utilisateurs et EAA, bonne ou mauvaise, pour utiliser cette information dans le calcul du niveau de confiance. Les attributs ne concernent actuellement que les sujets. Ils pourraient être étendus aux objets ou au contexte.

Concernant la dernière contribution, il serait intéressant d'étudier la possibilité d'un système de révocation automatique lors de changement de propriétaire.

Au delà de nos contributions, nous envisageons quatre directions pour nos futures recherche : l'amélioration de l'utilisation des ressources par la blockchain pour permettre aux objets connectés de s'y connecter sans l'usage d'une passerelle, l'amélioration du traitement des données des utilisateurs dans les produits de l'IdO, l'ajout de mécanismes de délégation dans les solutions de contrôle d'accès, et l'inclusion de mécanismes de révocation explicite des autorisations, c'est à dire de moyens de révoquer les permissions associées à un objet compromis sans attendre leurs expirations, et ce même dans le cas de solutions hors-ligne.



# Bibliography

- [Abdallah and Khayat, 2004] Abdallah, A. E. and Khayat, E. J. (2004). A formal model for parameterized role-based access control. In *IFIP World Computer Congress, TC 1*, pages 233–246. Springer. 84
- [Ahmad et al., 2018] Ahmad, T., Morelli, U., Ranise, S., and Zannone, N. (2018). A lazy approach to access control as a service (acaas) for iot: An aws case study. In *Proceedings of the 23rd ACM on Symposium on Access Control Models and Technologies*, pages 235–246. ACM. 80, 82, 101
- [Akl and Taylor, 1983] Akl, S. G. and Taylor, P. D. (1983). Cryptographic solution to a problem of access control in a hierarchy. *ACM Transactions on Computer Systems (TOCS)*, 1(3):239–248. 85
- [Ammar et al., 2018] Ammar, M., Russello, G., and Crispo, B. (2018). Internet of things: A survey on the security of iot frameworks. *Journal of Information Security and Applications*, 38:8–27. 64
- [Androulaki et al., 2018] Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., et al. (2018). Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference*, page 30. ACM. 30, 32
- [Anggorojati et al., 2012] Anggorojati, B., Mahalle, P. N., Prasad, N. R., and Prasad, R. (2012). Capability-based access control delegation model on the federated IoT network. In *Wireless Personal Multimedia Communications (WPMC), 2012 15th International Symposium on*, pages 604–608. IEEE. 86, 89
- [Antonakakis et al., 2017] Antonakakis, M., April, T., Bailey, M., Bernhard, M., Bursztein, E., Cochran, J., Durumeric, Z., Halderman, J. A., Invernizzi, L., Kallitsis, M., et al. (2017). Understanding the Mirai Botnet. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association. 7
- [Armando et al., 2005] Armando, A., Basin, D., Boichut, Y., Chevalier, Y., Compagna, L., Cuéllar, J., Drielsma, P. H., Héam, P.-C., Kouchnarenko, O., Mantovani, J., et al. (2005). The AVISPA tool for the automated validation of internet security protocols and applications. In *International conference on computer aided verification*, pages 281–285. Springer. 96
- [Ashibani et al., 2017] Ashibani, Y., Kauling, D., and Mahmoud, Q. H. (2017). A context-aware authentication framework for smart homes. In *Electrical and Computer Engineering (CCECE), 2017 IEEE 30th Canadian Conference on*, pages 1–5. IEEE. 78, 81

- [Atzori et al., 2012] Atzori, L., Iera, A., Morabito, G., and Nitti, M. (2012). The Social Internet of Things (SIoT)– when social networks meet the Internet of Things: Concept, architecture and network characterization. *Computer networks*, 56(16):3594–3608. 93
- [Bandara et al., 2016] Bandara, S., Yashiro, T., Koshizuka, N., and Sakamura, K. (2016). Access control framework for API-enabled devices in smart buildings. In *Communications (APCC), 2016 22nd Asia-Pacific Conference on*, pages 210–217. IEEE. 78, 81, 82
- [Barka et al., 2015] Barka, E., Mathew, S. S., and Atif, Y. (2015). Securing the Web of Things with Role-Based access control. In *International Conference on Codes, Cryptology, and Information Security*, pages 14–26. Springer. 84, 86
- [Bentov et al., 2014] Bentov, I., Lee, C., Mizrahi, A., and Rosenfeld, M. (2014). Proof of activity: Extending bitcoin’s proof of work via proof of stake. *ACM SIGMETRICS Performance Evaluation Review*, 42(3):34–37. 39
- [Bernabe et al., 2016] Bernabe, J. B., Hernández-Ramos, J. L., and Gomez, A. F. S. (2016). TACIoT: multidimensional trust-aware access control system for the Internet of Things. *Soft Computing*, 20(5):1763–1779. 91, 93, 112, 136, 137
- [Bertin et al., 2019] Bertin, E., Hussein, D., Sengul, C., and Frey, V. (2019). Access control in the internet of things: a survey of existing approaches and open research questions. *Annals of Telecommunications*, pages 1–14. 65
- [Burrows et al., 1990] Burrows, M., Abadi, M., and Needham, R. (1990). A logic of authentication. *ACM Transactions on Computer Systems*, 8:18–36. 85
- [Buterin, 2014] Buterin, V. (2014). Proof of stake : How i learned to love weak subjectivity. <https://blog.ethereum.org/2014/11/25/proof-stake-learned-love-weak-subjectivity>. Last Checked: July 28th, 2019. 37
- [Buterin et al., 2013] Buterin, V. et al. (2013). Ethereum white paper. <https://github.com/ethereum/wiki/wiki/White-Paper>. Last checked : 23/09/2016. 8, 24, 29
- [Castiglione et al., 2016] Castiglione, A., De Santis, A., Masucci, B., Palmieri, F., Castiglione, A., and Huang, X. (2016). Cryptographic hierarchical access control for dynamic structures. *IEEE Transactions on Information Forensics and Security*, 11(10):2349–2364. 85, 86
- [Cerf, 2015] Cerf, V. G. (2015). Access Control and the Internet of Things. *IEEE Internet Computing*, 19(5):96–c3. 90, 91, 137
- [Chen, 2008] Chen, L. (2008). Recommendation for key derivation using pseudorandom functions. Technical report, National Institute of Standards and Technology. 134
- [Cherkaoui et al., 2014] Cherkaoui, A., Bossuet, L., Seitz, L., Selander, G., and Borgaonkar, R. (2014). New Paradigms for Access Control in Constrained Environments. In *9th International Symposium on*

- 
- Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, page 4 p., Montpellier, France. 91, 92, 137
- [Christidis and Devetsikiotis, 2016] Christidis, K. and Devetsikiotis, M. (2016). Blockchains and smart contracts for the internet of things. *IEEE Access*, 4:2292–2303. 58
- [Decker and Wattenhofer, 2013] Decker, C. and Wattenhofer, R. (2013). Information propagation in the bitcoin network. In *IEEE International Conference on Peer-to-Peer Computing*, pages 1–10. 55
- [Dennis and Van Horn, 1966] Dennis, J. B. and Van Horn, E. C. (1966). Programming semantics for multiprogrammed computations. *Communications of the ACM*, 9(3):143–155. 67, 71
- [Dennis and Owen, 2015] Dennis, R. and Owen, G. (2015). Rep on the block: A next generation reputation system based on the blockchain. In *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 131–138. IEEE. 157
- [Dramé-Maigné et al., 2019] Dramé-Maigné, S., Laurent, M., and Castillo, L. (2019). Distributed Access Control Solution for the IoT based on Multi-endorsed Attributes and Smart Contracts. In *2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC)*, pages 1582–1587. IEEE. 142
- [Dramé-Maigné et al., 2018] Dramé-Maigné, S., Laurent, M., Castillo, L., and Ganem, H. (2018). Augmented chain of ownership: Configuring iot devices with the help of the blockchain. In *International Conference on Security and Privacy in Communication Systems*, pages 53–68. Springer. 166
- [Duffield and Diaz, 2015] Duffield, E. and Diaz, D. (2015). Dash : A privacy-centric crypto-currency. <https://www.dash.org/wp-content/uploads/2015/04/Dash-WhitepaperV1.pdf>. Last checked : 23/09/2016. 55
- [Eyal and Sirer, 2014] Eyal, I. and Sirer, E. G. (2014). Majority is not enough: Bitcoin mining is vulnerable. In *International Conference on Financial Cryptography and Data Security*, pages 436–454. Springer. 53
- [Fernández et al., 2017] Fernández, F., Alonso, Á., Marco, L., and Salvachúa, J. (2017). A model to enable application-scoped access control as a service for IoT using OAuth 2.0. In *Innovations in Clouds, Internet and Networks (ICIN), 2017 20th Conference on*, pages 322–324. IEEE. 78, 79, 81
- [Fernández-Caramés and Fraga-Lamas, 2018] Fernández-Caramés, T. M. and Fraga-Lamas, P. (2018). A review on the use of blockchain for the internet of things. *IEEE Access*. 58, 59, 60
- [Ferraiolo and Kuhn, 1992] Ferraiolo, D. F. and Kuhn, R. (1992). Role-based access controls. *15th NIST-NCSC National Computer Security Conference*, pages 554–563. 65, 67, 84
- [Fotiou et al., 2016] Fotiou, N., Kotsonis, T., Marias, G. F., and Polyzos, G. C. (2016). Access Control for the Internet of Things. In *Secure Internet of Things (SIoT), 2016 International Workshop on*, pages 29–38. IEEE. 95, 96

- [Garay et al., 2015] Garay, J., Kiayias, A., and Leonardos, N. (2015). The bitcoin backbone protocol: Analysis and applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 281–310. Springer. 26
- [Goyal et al., 2006] Goyal, V., Pandey, O., Sahai, A., and Waters, B. (2006). Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 89–98. Acm. 182
- [Gusmeroli et al., 2013] Gusmeroli, S., Piccione, S., and Rotondi, D. (2013). A capability-based security approach to manage access control in the Internet of Things. *Mathematical and Computer Modelling*, 58(5):1189–1205. 79, 82, 101
- [Hao et al., 2019] Hao, J., Huang, C., Ni, J., Rong, H., Xian, M., and Shen, X. S. (2019). Fine-grained data access control with attribute-hiding policy for cloud-based iot. *Computer Networks*. 79, 81
- [Hardt, 2012] Hardt, D. (2012). The OAuth 2.0 Authorization Framework. RFC 6749, RFC Editor. 65, 72
- [Harney and Muckenhirn, 1997] Harney, H. and Muckenhirn, C. (1997). Group key management protocol (gkmp) architecture. RFC 2094, RFC Editor. 182
- [Hemdi and Deters, 2016] Hemdi, M. and Deters, R. (2016). Using REST based protocol to enable ABAC within IoT systems. In *Information Technology, Electronics and Mobile Communication Conference (IEMCON), 2016 IEEE 7th Annual*, pages 1–7. IEEE. 91, 93, 137
- [Hernández-Ramos et al., 2013] Hernández-Ramos, J. L., Jara, A. J., Marín, L., and Skarmeta, A. F. (2013). Distributed capability-based access control for the internet of things. *Journal of Internet Services and Information Security (JISIS)*, 3(3/4):1–16. 91, 93, 112, 137
- [Hernández-Ramos et al., 2016] Hernández-Ramos, J. L., Jara, A. J., Marín, L., and Skarmeta Gómez, A. F. (2016). DCapBAC: embedding authorization logic into smart things through ECC optimizations. *International Journal of Computer Mathematics*, 93(2):345–366. 93
- [Hsiao et al., 2019] Hsiao, T.-C., Chen, T.-L., Chen, T.-S., and Chung, Y.-F. (2019). Elliptic curve cryptosystems-based date-constrained hierarchical key management scheme in internet of things. *Sensors and Materials*, 31(2):355–364. 85, 86
- [Hu et al., 2006] Hu, V. C., Ferraiolo, D., and Kuhn, D. R. (2006). *Assessment of access control systems*. US Department of Commerce, National Institute of Standards and Technology. 68
- [Hu et al., 2013] Hu, V. C., Ferraiolo, D., Kuhn, R., Friedman, A. R., Lang, A. J., Cogdell, M. M., Schnitzer, A., Sandlin, K., Miller, R., Scarfone, K., et al. (2013). Guide to attribute based access control (ABAC) definition and considerations. *NIST Special Publication*, 800(162). 65, 67, 69, 73, 143

- 
- [Hummen et al., 2014] Hummen, R., Shafagh, H., Raza, S., Voig, T., and Wehrle, K. (2014). Delegation-based Authentication and Authorization for the IP-based Internet of Things. In *Sensing, Communication, and Networking (SECON), 2014 Eleventh Annual IEEE International Conference on*, pages 284–292. Ieee. 80, 82
- [Hussein et al., 2017a] Hussein, D., Bertin, E., and Frey, V. (2017a). A Community-Driven Access Control Approach in Distributed IoT Environments. *IEEE Communications Magazine*, 55(3):146–153. 91, 93, 136, 137
- [Hussein et al., 2017b] Hussein, D., Bertin, E., and Frey, V. (2017b). Access control in IoT: From requirements to a candidate vision. In *Innovations in Clouds, Internet and Networks (ICIN), 2017 20th Conference on*, pages 328–330. IEEE. 112
- [INCITS, 2013] INCITS (2013). Next Generation Access Control - Functional Architecture (NGAC-FA). Standard INCITS 499-2013, American National Standard Institute. 80
- [INCITS, 2016] INCITS (2016). Next Generation Access Control - Generic Operations and Data Structures. Standard INCITS 526, American National Standards Institute. 80
- [Johnson et al., 2001] Johnson, D., Menezes, A., and Vanstone, S. (2001). The elliptic curve digital signature algorithm (ECDSA). *International Journal of Information Security*, 1(1):36–63. 79
- [Jones et al., 2015a] Jones, M., Bradley, J., and Sakimura, N. (2015a). JSON Web Signature (JWS). RFC 7515, RFC Editor. 72, 113
- [Jones et al., 2015b] Jones, M., Bradley, J., and Sakimura, N. (2015b). JSON Web Token (JWT). RFC 7519, RFC Editor. 72, 113
- [Jones et al., 2016] Jones, M., Bradley, J., and Tschofenig, H. (2016). Proof-of-possession key semantics for json web tokens (jwts). RFC 7800, RFC Editor. 135
- [Jones and Hildebrand, 2015] Jones, M. and Hildebrand, J. (2015). JSON Web Encryption (JWE). RFC 7516, RFC Editor. 72, 92, 113
- [Kaliski, 2000] Kaliski, B. (2000). PKCS# 5: Password-based cryptography specification version 2.0. RFC 2898, RFC Editor. 134
- [Khan and Salah, 2018] Khan, M. A. and Salah, K. (2018). Iot security: Review, blockchain solutions, and open challenges. *Future Generation Computer Systems*, 82:395–411. 65
- [Kouicem et al., 2018] Kouicem, D. E., Bouabdallah, A., and Lakhlef, H. (2018). Internet of things security: A top-down survey. *Computer Networks*, 141:199–221. 65
- [Kwon, 2014] Kwon, J. (2014). Tendermint: Consensus without mining. <https://www.weusecoins.com/assets/pdf/library/TendermintConsensuswithoutMining.pdf>. 36, 37

- [Lin and Liao, 2017] Lin, I.-C. and Liao, T.-C. (2017). A survey of blockchain security issues and challenges. *IJ Network Security*, 19(5):653–659. 175
- [Liu et al., 2017] Liu, Q., Zhang, H., Wan, J., and Chen, X. (2017). An Access Control Model for Resource Sharing based on the Role-Based Access Control Intended for Multi-domain Manufacturing Internet of Things. *IEEE Access*. 88, 89
- [Maler et al., 2015] Maler, E., Machulak, M., and Catalano, D. (2015). User-Managed Access (UMA) Profile of OAuth 2.0. Standard, Kantara Initiative. Last accessed: January 5, 2018. 65
- [Mathew, 2013] Mathew, S. S. (2013). *Classifying and clustering the web of things*. PhD thesis, University of Adelaide. 84
- [Mathew et al., 2011] Mathew, S. S., Atif, Y., Sheng, Q. Z., and Maamar, Z. (2011). Web of things: Description, discovery and integration. In *Internet of Things (iThings/CPSCoM), 2011 International Conference on and 4th International Conference on Cyber, Physical and Social Computing*, pages 9–15. IEEE. 84
- [McCorry et al., 2017] McCorry, P., Heilman, E., and Miller, A. (2017). Atomically trading with roger: Gambling on the success of a hardfork. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 334–353. Springer. 46
- [Menezes et al., 1996] Menezes, A. J., Van Oorschot, P. C., and Vanstone, S. A. (1996). *Handbook of applied cryptography*. CRC press. 16
- [Merkle, 1980] Merkle, R. C. (1980). Protocols for public key cryptosystems. In *IEEE Symposium on Security and privacy*, volume 122. 16, 19
- [Miers et al., 2013] Miers, I., Garman, C., Green, M., and Rubin, A. D. (2013). Zerocoin: Anonymous distributed e-cash from bitcoin. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 397–411. IEEE. 54
- [Moses et al., 2005] Moses, T. et al. (2005). Extensible access control markup language (XACML) version 2.0. Standard, OASIS. 73, 92
- [Nakamoto, 2008] Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. 15, 27, 50
- [Neuman and Ts'o, 1994] Neuman, B. C. and Ts'o, T. (1994). Kerberos: An authentication service for computer networks. *IEEE Communications magazine*, 32(9):33–38. 79
- [Ning et al., 2015] Ning, H., Liu, H., and Yang, L. T. (2015). Aggregated-proof based hierarchical authentication scheme for the internet of things. *IEEE Transactions on Parallel and Distributed Systems*, 26(3):657–667. 84, 86
- [NIST, 2001] NIST, U. (2001). Descriptions of SHA-256, SHA-384, and SHA-512. <https://web.archive.org/web/20130526224224/http://csrc.nist.gov/groups/STM/cavp/documents/shs/sha256-384-512.pdf>. 134



- 
- [Novo, 2018] Novo, O. (2018). Blockchain meets iot: an architecture for scalable access management in iot. *IEEE Internet of Things Journal*. 98, 99, 100, 163
- [Osaka et al., 2008] Osaka, K., Takagi, T., Yamazaki, K., and Takahashi, O. (2008). An efficient and secure rfid security method with ownership transfer. In *RFID security*, pages 147–176. Springer. 8
- [Ouaddah et al., 2016a] Ouaddah, A., Abou Elkalam, A., and Ait Ouahman, A. (2016a). FairAccess: a new Blockchain-based access control framework for the Internet of Things. *Security and Communication Networks*, 9(18):5943–5964. 98, 99, 162, 163
- [Ouaddah et al., 2017a] Ouaddah, A., Elkalam, A. A., and Ouahman, A. A. (2017a). Towards a Novel Privacy-Preserving Access Control Model Based on Blockchain Technology in IoT. In *Europe and MENA Cooperation Advances in Information and Communication Technologies*, pages 523–533. Springer. 98, 99, 163
- [Ouaddah et al., 2016b] Ouaddah, A., Mousannif, H., Elkalam, A. A., and Ouahman, A. A. (2016b). Access control in iot: Survey & state of the art. In *Multimedia Computing and Systems (ICMCS), 2016 5th International Conference on*, pages 272–277. IEEE. 192
- [Ouaddah et al., 2017b] Ouaddah, A., Mousannif, H., Elkalam, A. A., and Ouahman, A. A. (2017b). Access control in the Internet of Things: Big challenges and new opportunities. *Computer Networks*, 112:237–262. 65, 207
- [Ouaddah et al., 2015] Ouaddah, A., Mousannif, H., and Ouahman, A. A. (2015). Access control models in IoT: The road ahead. In *Computer Systems and Applications (AICCSA), 2015 IEEE/ACS 12th International Conference of*, pages 1–2. IEEE. 64
- [Pappu et al., 2002] Pappu, R., Recht, B., Taylor, J., and Gershenfeld, N. (2002). Physical one-way functions. *Science*, 297(5589):2026–2030. 92
- [Park and Sandhu, 2004] Park, J. and Sandhu, R. (2004). The UCON ABC usage control model. *ACM Transactions on Information and System Security (TISSEC)*, 7(1):128–174. 65
- [Pass et al., 2017] Pass, R., Seeman, L., and Shelat, A. (2017). Analysis of the blockchain protocol in asynchronous networks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 643–673. Springer. 26, 145, 146, 167
- [Patel et al., 2016] Patel, S., Patel, D. R., and Navik, A. P. (2016). Energy efficient integrated authentication and access control mechanisms for Internet of Things. In *Internet of Things and Applications (IOTA), International Conference on*, pages 304–309. IEEE. 95, 96
- [Pinno et al., 2017] Pinno, O. J. A., Gregio, A. R. A., and De Bona, L. C. (2017). Controlchain: Blockchain as a central enabler for access control authorizations in the iot. In *GLOBECOM 2017-2017 IEEE Global Communications Conference*, pages 1–6. IEEE. 98, 99, 162, 163
- [Popov, 2016] Popov, S. (2016). The tangle [iota whitepaper]. *cit. on*, page 131. 194

- [Qiu et al., 1985] Qiu, L., Zhang, Y., Wang, F., Kyung, M., and Mahajan, H. R. (1985). Trusted computer system evaluation criteria. Standard DoD 5200.28-STD, National Computer Security Center. [66](#)
- [Ray et al., 2018] Ray, B. R., Abawajy, J., Chowdhury, M., and Alelaiwi, A. (2018). Universal and secure object ownership transfer protocol for the internet of things. *Future Generation Computer Systems*, 78:838–849. [8](#)
- [Ray et al., 2017] Ray, I., Alangot, B., Nair, S., and Achuthan, K. (2017). Using attribute-based access control for remote healthcare monitoring. In *Software Defined Systems (SDS), 2017 Fourth International Conference on*, pages 137–142. IEEE. [80](#), [82](#)
- [Reid and Harrigan, 2013] Reid, F. and Harrigan, M. (2013). An analysis of anonymity in the bitcoin system. In *Security and privacy in social networks*, pages 197–223. Springer. [54](#), [100](#)
- [Rekleitis et al., 2014] Rekleitis, E., Rizomiliotis, P., and Gritzalis, S. (2014). How to protect security and privacy in the iot: a policy-based rfid tag management protocol. *Security and Communication Networks*, 7(12):2669–2683. [8](#)
- [Rescorla and Modadugu, 2012] Rescorla, E. and Modadugu, N. (2012). Datagram transport layer security version 1.2. RFC 6347, RFC Editor. [80](#)
- [Roman et al., 2013] Roman, R., Zhou, J., and Lopez, J. (2013). On the features and challenges of security and privacy in distributed internet of things. *Computer Networks*, 57(10):2266–2279. [64](#), [65](#), [207](#)
- [Rosenfeld, 2012] Rosenfeld, M. (2012). Overview of colored coins. *White paper; bitcoil. co. il*, page 41. [8](#), [166](#)
- [Rotondi et al., 2011] Rotondi, D., Seccia, C., and Piccione, S. (2011). Access control & iot: Capability based authorization access control system. In *1st IoT International Forum*. [79](#), [82](#)
- [Saadeh et al., 2018] Saadeh, M., Sleit, A., Sabri, K. E., and Almobaideen, W. (2018). Hierarchical architecture and protocol for mobile object authentication in the context of iot smart cities. *Journal of Network and Computer Applications*, 121:1–19. [88](#), [89](#), [90](#), [101](#)
- [Sain et al., 2017] Sain, M., Kang, Y. J., and Lee, H. J. (2017). Survey on security in internet of things: State of the art and challenges. In *Advanced Communication Technology (ICACT), 2017 19th International Conference on*, pages 699–704. IEEE. [64](#)
- [Saltzer and Schroeder, 1975] Saltzer, J. H. and Schroeder, M. D. (1975). The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308. [71](#)
- [Samarati and de Vimercati, 2000] Samarati, P. and de Vimercati, S. C. (2000). Access control: Policies, models, and mechanisms. In *International School on Foundations of Security Analysis and Design*, pages 137–196. Springer. [66](#), [67](#)

- 
- [Sandhu, 2000] Sandhu, R. (2000). Engineering authority and trust in cyberspace: The OM-AM and RBAC way. In *Proceedings of the fifth ACM workshop on Role-based access control*, pages 111–119. ACM. 65
- [Schaub et al., 2016] Schaub, A., Bazin, R., Hasan, O., and Brunie, L. (2016). A trustless privacy-preserving reputation system. In *IFIP International Information Security and Privacy Conference*, pages 398–411. Springer. 157
- [Schneider, 2003] Schneider, F. B. (2003). Least privilege and more [computer security]. *IEEE Security & Privacy*, 99(5):55–59. 88
- [Schwartz et al., 2014] Schwartz, D., Youngs, N., and Britto, A. (2014). The ripple protocol consensus algorithm. *Ripple Labs White Paper*. 39, 40
- [Sciancalepore et al., 2018] Sciancalepore, S., Piro, G., Caldarola, D., Boggia, G., and Bianchi, G. (2018). On the design of a decentralized and multiauthority access control scheme in federated and cloud-assisted cyber-physical systems. *IEEE Internet of Things Journal*, 5(6):5190–5204. 89, 101
- [Seitz et al., 2013] Seitz, L., Selander, G., and Gehrman, C. (2013). Authorization framework for the internet-of-things. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2013 IEEE 14th International Symposium and Workshops on a*, pages 1–6. IEEE. 91, 92, 93, 137
- [Shelby et al., 2014] Shelby, Z., Hartke, K., and Bormann, C. (2014). The constrained application protocol (CoAP). RFC 7252, RFC Editor. 79, 93
- [Shirey, 2007] Shirey, R. W. (2007). Internet security glossary. RFC 4949, IETF. 66
- [Sicari et al., 2015] Sicari, S., Rizzardi, A., Grieco, L. A., and Coen-Porisini, A. (2015). Security, privacy and trust in internet of things: The road ahead. *Computer Networks*, 76:146–164. 64
- [Sompolinsky and Zohar, 2015] Sompolinsky, Y. and Zohar, A. (2015). Secure high-rate transaction processing in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 507–527. Springer. 26
- [Szabo, 1997] Szabo, N. (1997). Formalizing and securing relationships on public networks. *First Monday*, 2(9). 8, 24
- [Tamboli and Dambawade, 2016] Tamboli, M. B. and Dambawade, D. (2016). Secure and efficient CoAP based authentication and access control for Internet of Things (IoT). In *Recent Trends in Electronics, Information & Communication Technology (RTEICT), IEEE International Conference on*, pages 1245–1250. IEEE. 79, 82
- [Tourani et al., 2016] Tourani, R., Mick, T., Misra, S., and Panwar, G. (2016). Security, privacy, and access control in information-centric networking: A survey. *arXiv preprint arXiv:1603.03409*. 64

- [Uriarte et al., 2016] Uriarte, M., López, O., Blasi, J., Lázaro, O., González, A., Prada, I., Olivares, E., Palau, C. E., Molina, B., Portugués, M. A., et al. (2016). Sensing enabled capabilities for access control management: Iot as an enabler for the advanced management of access control. In *Internet-of-Things Design and Implementation (IoTDI), 2016 IEEE First International Conference on*, pages 253–258. IEEE. 88, 89
- [Van Tilborg, 2014] Van Tilborg, H. C., editor (2014). *Encyclopedia of cryptography and security*. Springer Science & Business Media. 67, 71
- [Vukolić, 2015] Vukolić, M. (2015). The quest for scalable blockchain fabric: Proof-of-work vs. bft replication. In *International workshop on open problems in network security*, pages 112–125. Springer. 33
- [Westerinen et al., 2001] Westerinen, A., Schnizlein, J., Strassner, J., Scherling, M., Quinn, B., Herzog, S., Huynh, A., Carlson, M., Perry, J., and Waldbusser, S. (2001). Terminology for policy-based management. RFC 3198, RFC Editor. 69
- [Wood, 2014] Wood, G. (2014). Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*. 8, 29
- [Wu et al., 2017] Wu, X., Steinfeld, R., Liu, J., and Rudolph, C. (2017). An implementation of access-control protocol for IoT home scenario. In *Computer and Information Science (ICIS), 2017 IEEE/ACIS 16th International Conference on*, pages 31–37. IEEE. 95, 115
- [Xu et al., 2018a] Xu, R., Chen, Y., Blasch, E., and Chen, G. (2018a). Blendcac: A smart contract enabled decentralized capability-based access control mechanism for iot. *Computers*, 7(3):39. 98, 99, 163
- [Xu et al., 2018b] Xu, R., Chen, Y., Blasch, E., and Chen, G. (2018b). A federated capability-based access control mechanism for internet of things (iots). In *Sensors and Systems for Space Applications XI*, volume 10641, page 106410U. International Society for Optics and Photonics. 87, 89, 90, 101
- [Yan et al., 2019] Yan, H., Wang, Y., Jia, C., Li, J., Xiang, Y., and Pedrycz, W. (2019). Iot-fbac: Function-based access control scheme using identity-based encryption in iot. *Future Generation Computer Systems*. 79, 81
- [Yang et al., 2017] Yang, Y., Wu, L., Yin, G., Li, L., and Zhao, H. (2017). A Survey on Security and Privacy Issues in Internet-of-Things. *IEEE Internet of Things Journal*. 64
- [Yavatkar et al., 2000] Yavatkar, R., Pendarakis, D., and Guerin, R. (2000). A framework for policy-based admission control. RFC 2753, RFC Editor. 69
- [Zhang et al., 2018a] Zhang, P., Liu, J. K., Yu, F. R., Sookhak, M., Au, M. H., and Luo, X. (2018a). A survey on access control in fog computing. *IEEE Communications Magazine*, 56(2):144–149. 65

- 
- [Zhang et al., 2018b] Zhang, Y., Kasahara, S., Shen, Y., Jiang, X., and Wan, J. (2018b). Smart Contract-Based Access Control for the Internet of Things. *arXiv preprint arXiv:1802.04410*. 98, 99, 162, 163, 192
- [Zyskind et al., 2015] Zyskind, G., Nathan, O., and Pentland, A. (2015). Decentralizing privacy: Using blockchain to protect personal data. In *Security and Privacy Workshops (SPW), 2015 IEEE*, pages 180–184. 98, 163

**Titre :** Blockchain et contrôle d'accès : Vers un Internet des Objets plus sécurisé

**Mots clés :** Internet des Objets, Contrôle d'accès, Blockchain, Contrôle d'accès à base d'attributs, Contrôle d'accès à base de jetons, Traçabilité de propriétaires

**Résumé :** L'Internet des Objets (IdO) a besoin d'améliorer sa sécurité pour assurer l'expansion du domaine et son adoption par les utilisateurs. Mais le manque d'espace, de puissance de calcul, ou encore l'accès au réseau limite l'utilisation des méthodes traditionnelles. De nouvelles méthodes doivent donc être adoptées.

Le contrôle d'accès est une étape importante vers la sécurisation d'un système. Il permet de restreindre l'accès aux informations confidentielles et aux fonctions sensibles. Cette thèse propose quatre contributions dont trois sont liées au contrôle d'accès.

Tout d'abord, nous examinons l'état de l'art pour déterminer l'influence de l'architecture sur les propriétés d'une solution et proposer une taxonomie du contrôle d'accès dans l'IdO.

Ensuite, nous proposons un ensemble de bibliothèques pour aider les développeurs à intégrer les mécanismes de contrôle d'accès dans

leurs produits. Ces bibliothèques gèrent l'émission, le stockage, et la vérification de jetons matérialisant l'autorisation.

La blockchain est un registre distribué qui enregistre et ordonne les transactions. Elle est utilisée dans notre troisième contribution pour gérer les attributs des utilisateurs, les politiques de contrôle d'accès, et la confiance entre les différentes entités.

Notre quatrième contribution élargit le spectre de la sécurité de l'IdO et se concentre sur les propriétaires. Au cours de sa vie, un *objet* est susceptible de changer de main. Nous fournissons un système de suivi des propriétaires ainsi qu'une preuve de propriété indépendante. De plus, nous proposons un système de gestion des secrets liés aux objets et un mécanisme de publication des propriétés dynamiques de ces objets qui pourraient intéresser des utilisateurs potentiels.

**Title:** Blockchain and access control: Towards a more secure Internet of Things

**Keywords:** Internet of Things, Access control, Blockchain, ABAC, CapBAC, Ownership tracking

**Abstract:** The Internet of Things (IoT) needs better security if we want to ensure the expansion of the field and user adoption. But the lack of memory, of computational power, or access to the network limit the use of traditional methods. New ones must therefore be devised.

Access control is an important step towards securing a system. It restricts access to private information and sensitive functions. This thesis proposes four contributions, among which three are related to access control.

First, we survey the state of the art to determine the influence of architecture on the properties of a solution and propose a taxonomy of IoT access control.

Second, we propose a set of libraries to help

developers integrate access control mechanisms into their products. These libraries handle the issuance, storing, and verification of tokens that materialize authorization.

The blockchain is a distributed ledger that registers and orders transactions. It is used in our third contribution to manage user's attributes, access control policies, and trust.

Our fourth contribution broadens the specter of IoT security and focuses on device ownership.

Throughout its life, an IoT device is likely to change hands. We provide an ownership tracking system as well as an independent proof of ownership. Additionally, we propose a management system for device-related secret and a mechanism for publishing dynamic device properties that might interest potential users.

