



**João Pedro Ramos
Rebelo de Amaral**

**Autenticação em interações com dispositivos IoT
Authentication in interactions with IoT devices**



**João Pedro Ramos
Rebello de Amaral**

Autenticação em interações com dispositivos IoT
Authentication in interactions with IoT devices

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor André Zúquete, Professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor Hélder Gomes (co-orientador), Professor adjunto da Escola Superior de Tecnologia e Gestão de Águeda da Universidade de Aveiro.

o júri / the jury

presidente / president

Prof. Doutor Joaquim João Estrela Ribeiro Silvestre Madeira
Professor Auxiliar da Universidade de Aveiro

vogais / examiners committee

Prof. Doutor Pedro Miguel Alves Brandão
Professor Auxiliar da Faculdade de Ciências da Universidade do Porto (arguente principal)

Prof. Doutor André Ventura da Cruz Marnoto Zúquete
Professor Auxiliar da Universidade de Aveiro (orientador)

agradecimentos / acknowledgements

Em primeiro lugar gostaria de agradecer ao Professor André Zúquete e ao Professor Hélder Gomes, tanto pela oportunidade de trabalhar sobre a sua orientação, assim como por todo o apoio e disponibilidade prestada ao longo da realização desta dissertação.

Agradeço também a todos os meus colegas que me acompanharam nestes últimos cinco anos académicos por todo o convívio, amizade e experiências partilhadas.

Por fim, um agradecimento especial aos meus pais, aos meus irmãos e a minha avó que sempre me apoiaram e motivaram ao longo deste percurso académico.

Palavras Chave

Internet das Coisas, Segurança, Privacidade, Autenticação, Autorização, Controlo de acesso, Comunicação segura

Resumo

A Internet das Coisas (IdC) visa fornecer aos dispositivos do quotidiano com recursos computacionais e de comunicação, desta forma integrando-os à actual e amplamente disponível Internet. No entanto, para que essa visão prospere, é importante garantir a comunicação segura com os dispositivos e os dados gerados pelos mesmos, o que geralmente é difícil dadas as restrições de recursos que muitos desses dispositivos apresentam, assim como devido à natureza distribuída da IdC. Esta dissertação começa apresentando uma visão geral dos conceitos que envolvem o domínio da IdC, descrevendo alguns dos principais problemas que enfrenta, principalmente em relação à privacidade e segurança. De seguida é apresentada uma proposta de uma nova arquitetura para proteger estes ambientes, através da descentralização das entidades de comunicação e da definição de um mecanismo de controlo de acesso dividido, usado no estabelecimento seguro de sessões. Esta arquitetura é então implementada como um protótipo físico usando periféricos e tecnologias comuns na IdC, juntamente com a sua avaliação numa perspectiva de segurança.

Keywords

Internet of Things, Security, Privacy, Authentication, Authorization, Access control, Secure communication

Abstract

The Internet of Things aims to provide everyday devices with computing and communication capabilities, ultimately integrating them within the current widely available Internet. However, for this vision to thrive, it is important to assure secure communication with the devices and their generated data, which is often difficult given the resource constraints presented by many devices and the overall distributed nature of the IoT. This dissertation starts by presenting an overview of the concepts involving the IoT domain while describing some of the major issues it faces, namely regarding privacy and security. It is then followed by the presentation of a novel architecture approach for securing such environments, through the decentralization of the communication entities and the definition of a split access control mechanism for secure session establishment. The developed architecture is then implemented as a physical prototype using common IoT peripherals and technologies, along with its evaluation from a security perspective.

Contents

Contents	i
List of Figures	v
List of Tables	vii
Acronyms	ix
1 Introduction	1
1.1 Objectives	2
1.2 Dissertation outline	3
2 The Internet of Things	5
2.1 IoT rise, concept and vision	5
2.2 Defining the Internet of Things (IoT)	8
2.3 Wireless Sensor Network	9
2.4 Cloud Computing	10
2.5 Application areas	11
2.5.1 Personal and Home	11
2.5.2 Enterprise	12
2.5.3 Utilities	12
2.5.4 Mobile	12
2.6 Understanding the IoT Architecture	13
2.6.1 Software architecture	14
2.6.2 The four layer IoT architecture	16
2.6.3 Hardware architecture	17
2.7 IoT enabling technologies	18
2.7.1 Radio Frequency Identification (Radio-Frequency Identification (RFID)) . . .	18
2.7.2 Near Field Communication (Near-Field Communication (NFC))	19
2.7.3 Wi-Fi	19

2.7.4	Bluetooth	19
2.7.5	ZigBee	20
2.7.6	Datagram Transport Layer Security (DTLS)	21
2.7.7	Constrained Application Protocol (CoAP)	21
2.7.8	Message Queue Telemetry Transport (MQTT)	21
2.8	IoT problems and challenges	22
2.8.1	Standardization	22
2.8.2	Addressing and networking	23
2.8.3	Security and privacy	24
3	IoT Security overview	25
3.1	IoT security requirements	26
3.2	IoT security threats / vulnerabilities	28
3.3	Architectural layers security	30
3.3.1	Perception layer security	31
3.3.2	Network layer security	32
3.3.3	Application layer security	32
3.4	IoT Framework architectures and security	33
3.4.1	Azure IoT Suite	34
3.4.2	Amazon Web Services (AWS) IoT	35
3.5	Blockchain security solutions for IoT	37
4	Authentication in IoT	39
4.1	Cryptography fundamentals for authentication	39
4.1.1	Symmetric key cryptography	40
4.1.2	Asymmetric key cryptography	40
4.1.3	Cryptographic Hash Functions	42
4.1.4	Message Authentication Code (MAC)	42
4.1.5	Digital certification	43
4.2	Authentication principles and classification	43
4.2.1	Authentication classification in the IoT	44
4.3	Authentication protocols	46
4.3.1	Kerberos	46
4.3.2	OAuth 2.0	48
4.3.3	eXtensible Access Control Markup Language (XACML)	49
4.3.4	Lightweight cryptography solutions for IoT	50
5	IoT Security Architecture	51
5.1	Towards a first architecture draft	51

5.1.1	Desired security objectives	51
5.1.2	The initial architecture draft	52
5.1.3	Towards a more reliable and robust security architecture	53
5.2	Final architecture specification	54
5.2.1	IoT devices	54
5.2.2	Device Hosts (Device Hosts (DHs))	55
5.2.3	Device Drivers (Device Drivers (DDs))	56
5.2.4	IoT Gateways	56
5.2.5	IoT Authentication, Authorization and Accounting (AAA) Controller (A3C) .	57
5.2.6	Device Host Manager (DHM)	58
5.2.7	Client	58
5.3	Architecture authentication process	58
5.3.1	Configuration/Authorization Ticket structure	58
5.3.2	Network bootstrap and configuration	59
5.3.3	Identification and naming	60
5.3.4	Ticket fetching protocol	61
5.3.5	Session setup protocol	62
5.3.6	Client-DD session establishment	63
6	Prototype implementation	65
6.1	The physical prototype and implemented technologies	65
6.1.1	Web server framework - Tornado	65
6.1.2	Persistence frameworks/libraries	66
6.1.3	MQTT broker - Mosquitto	66
6.1.4	Bluetooth Low Energy (BLE)	67
6.1.5	Data serialization - Protocol Buffers	70
6.1.6	Cryptography	70
6.2	Entities' life cycle and processing workflows	71
6.2.1	Device Host (DH)	71
6.2.2	Gateway (GW)	73
6.2.3	Device Host Manager (Device Host Manager (DHM))	74
6.2.4	Authentication, Authorization and Accounting Controller (A3C) server	74
6.2.5	IoT devices	74
6.2.6	Device Driver (DD)	75
6.2.7	Client	75
7	Security evaluation and results	77
7.1	Security evaluation of the theoretical model and its underlying protocols	77

7.1.1	Device spoofing and impersonation	78
7.1.2	Ticket stealing / Man-in-the-middle (Man-In-The-Middle (MITM))	79
7.1.3	Privacy and enumeration	80
7.2	Security evaluation regarding the prototype and adopted technologies	80
7.2.1	Bluetooth Low Energy (BLE) security overview	81
7.3	Performance of cryptographic algorithms	83
7.3.1	DH	84
7.3.2	Gateway (GW)	85
7.3.3	Client, A3Cs and DHM	85
8	Conclusions	87
8.1	Final considerations	87
8.2	Future work	88
	References	89
A	Protocol buffer configuration file	97
B	A3C GW debug log	99
C	A3C DD debug log	101
D	DHM debug log	103
E	DH debug log	105
F	GW debug log	107
G	Client debug log	111
H	DH internal states	113

List of Figures

1.1	Technological and social aspects of IoT [88]	2
2.1	Internet evolution towards IoT [73]	6
2.2	Global IoT revenue by technology segment (\$bn) 2018-2023 [36]	8
2.3	Sensor nodes scattered in a sensor field [5]	10
2.4	Conceptual IoT framework with Cloud Computing at the centre [45]	11
2.5	General stages of an IoT application [117]	13
2.6	Topology of a generic IoT network (adapted from [142])	14
2.7	Functionality perspective of Service-oriented Architecture (SOA) four layer IoT architecture [27]	16
2.8	General architecture of an IoT embedded device [117]	18
2.9	Technologies associated with IoT [27]	18
3.1	The CIA triad [107]	27
3.2	Azure IoT Architecture (adapted from [10])	35
3.3	Amazon Web Services IoT Architecture [105]	37
4.1	Symmetric key encryption [124]	40
4.2	Asymmetric key encryption [124]	41
4.3	Hashing algorithm [95]	42
4.4	Taxonomy of IoT authentication schemes [47]	46
4.5	Kerberos workflow (adapted from [17])	47
4.6	Kerberos authenticator structure (left) and ticket (right)	48
4.7	OAuth 2.0 Protocol Flow (adapted from [49])	49
4.8	XACML architecture and message flow [16]	50
5.1	Initial draft of the IoT architecture	53
5.2	Developed IoT Security Architecture	55
5.3	Network bootstrap and configuration protocol	60
5.4	Ticket fetching protocol	62

5.5	Session setup protocol	63
5.6	Client-DD session establishment (not considering discovery protocols)	64
6.1	BLE advertisement [11]	67
6.2	BLE connection [11]	68
6.3	BLE architectural layers [21]	68
6.4	Internal structure of the DH BLE server.	72

List of Tables

2.1	Open IoT research issues (adapted from [14])	23
3.1	Security issues and attacks affecting IoT architectural layers [151])	33
6.1	Programming libraries, used for each module and entity	71
6.2	Gateway REST API specification.	73
6.3	A3C REST API specification.	74
6.4	DDs API specification.	75
7.1	ESP32 performance comparison between RSA and different ECDSA curves	84
7.2	Raspberry pi performance comparison between RSA and different ECDSA curves	85
7.3	Laptop performance comparison between RSA and different ECDSA curves	85
H.1	DH internal states.	113

Acronyms

IoT	Internet of Things	RF	Radio Frequency
PKI	Public Key Infrastructure	WPAN	Wide Personal Area Network
ARPANET	Advanced Research Projects Agency Network	SIG	Special Interest Group
TCP	Transmission Control Protocol	BLE	Bluetooth Low Energy
IP	Internet Protocol	L2CAP	Logical Link Control and Adaptation Protocol
ISP	Internet Service Provider	GAP	Generic Access Profile
WWW	World Wide Web	ATT	Attribute Protocol
NFC	Near-Field Communication	GATT	Generic Access Profile
RFID	Radio-Frequency Identification	ZC	ZigBee Coordinator
M2M	Machine to Machine	ZR	ZigBee Router
M2H	Machine to Human	ZER	ZigBee End Device
MIT	Massachusetts Institute of Technology	DTLS	Datagram Transport Layer Security
API	Application Programming Interface	CoAP	Constrained Application Protocol
IEEE	Institute of Electrical and Electronics Engineers	UDP	User Datagram Protocol
IETF	Internet Engineering Task Force	CoRE	Constrained RESTful Environments
NIST	National Institute of Standards and Technology	MQTT	Message Queue Telemetry Transport
W3C	World Wide Web Consortium	SSL	Secure Socket Layer
CPS	Cyber-Physical System	ONS	Object Naming Services
HTTP	HyperText Transfer Protocol	IPv4	Internet Protocol version 4
REST	Representational State Transfer	IPv6	Internet Protocol version 6
WSN	Wireless Sensor Network	6LoWPAN	IPv6 over Low-Power Wireless Personal Area Networks
IaaS	Infrastructure as a Service	URN	Uniform Resource Name
PaaS	Platform as a Service	uID	Ubiquitous ID
SaaS	Software as a Service	NDN	Named Data Networking
QoS	Quality of Service	CAGR	Compound Annual Growth Rate
SOA	Service-oriented Architecture	IT	Information technology
ITU	International Telecommunication Union	DoS	Denial-of-Service
SSAS	Service Support and Application Support	GSMA	GSM Association
GPS	Global Positioning System	UART	Universal Asynchronous Receiver-Transmitter
UUID	Universal Unique Identifier	RSA	Rivest–Shamir–Adleman
IFP	Interface Profile	ECC	Elliptic-Curve Cryptography
SoC	System on a chip	MITM	Man-In-The-Middle
		RAM	Random-Access Memory
		OWASP	Open Web Application Security Project

IPsec	Internet Protocol Security	DD	Device Driver
WPKI	Wireless Public Key Infrastructure	DH	Device Host
SCA	Side Channel Attack	A3C	Authentication, Authorization and Accounting Controller
PAN	Personal Area Network	DHM	Device Host Manager
AMQP	Advanced Message Queuing Protocol	I2C	Inter-Integrated Circuit
AWS	Amazon Web Services	SPI	Serial Peripheral Interface
IAM	Identity and Access Management	FIFO	First In First Out
FS	Forward Secrecy	ID	Identifier
TLS	Transport Layer Security	GW	Gateway
XOR	Exclusive Or	PBKDF2	Password-Based Key Derivation Function 2
ECB	Electronic Codebook	URI	Universal Resource Identifier
CBC	Cipher Block Chaining	I/O	Input/Output
CA	Certification Authority	FAT	File Allocation Table
MAC	Message Authentication Code	OAEP	Optimal Asymmetric Encryption Padding
RFC	Request for Comments	PSS	Probabilistic Signature Scheme
XACML	eXtensible Access Control Markup Language	PKCS	Public-Key Cryptography Standards
PUF	Physical Unclonable Function	AES	Advanced Encryption Standard
TRNG	True Random Number Generator	HMAC	Hash-based Message Authentication Code
TPM	Trusted Platform Module	SHA	Secure Hash Algorithm
KDC	Key Distribution Center	CCM	Counter mode with CBC-MAC
TGS	Ticket Granting Service	SSH	Secure Shell
PDP	Policy Decision Point	IKE	Internet Key Exchange
PEP	Policy Enforcement Point	ECDH	Elliptic Curve Diffie Hellman
PAP	Policy Administration Point	ECDSA	Elliptic Curve Digital Signature Algorithm
PIP	Policy Information Point		
PRP	Policy Retrieval Point		
AAA	Authentication, Authorization and Accounting		

Introduction

Ever since its conception, the Internet has been constantly expanding and maturing, and has now reached what is considered its next evolutionary step: the IoT. Although it has no unique and consensual definition, the IoT is a paradigm where everyday objects can be equipped with identifying, sensing, networking, acting and processing capabilities, that will ultimately allow them to communicate over the Internet in order to fulfill a certain objective [29]. These sensors will be present in our homes, workplaces, cars and other devices not usually connected to the Internet, revolutionizing the way we interact with the digital and physical world.

This way, the IoT has caught over the last decade the attention of society, industry and academy as a way of technologically enhancing daily activities, leading to the creation of new business models, products and services, while representing a broad source of research topics and ideas [55].

Due to its wide variety of applicational areas and use cases, it resulted in brand new and profitable market, which quickly led to the exponential growth of the number of IoT composing devices. According to companies like Gartner, more than 20 billion of IoT devices are predicted to be connected to the Internet by 2020 [37]

However, as companies race to get IoT devices to market, many are forgetting about security or consider it to be an afterthought. The growing presence of these devices enables new attack methods and new attack surfaces for criminals and hackers to exploit, posing serious security and privacy issues. Numerous attacks on IoT devices have already been demonstrated, and these attacks could have significant resulting consequences for all Internet participants. Therefore, it is vital that companies and developers take the time to consider the security of their devices and adopt appropriate security solutions. One other emerging concern comes from the fact that most of the IoT embedded devices are low-cost, thus having reduced processing power and available memory. This originates a series of new problems since most security related operations are computationally expensive, and general security adopted methodologies, such as public key cryptography algorithms and Public Key Infrastructure (PKI) may not be feasible to implement [96].

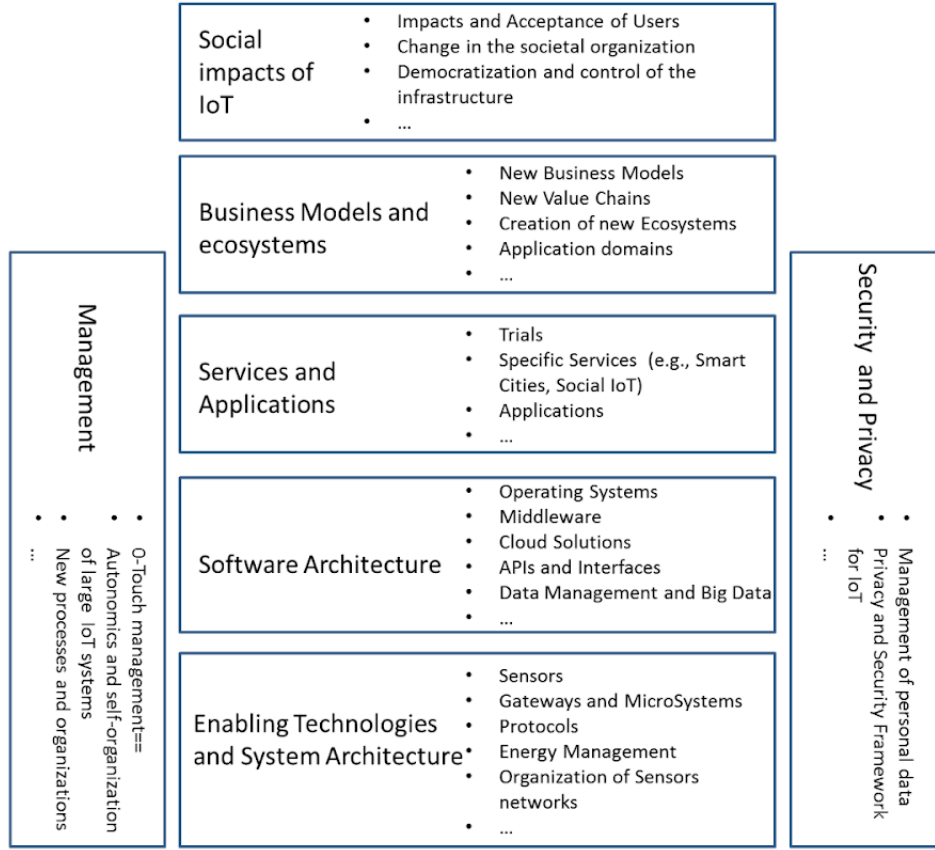


Figure 1.1: Technological and social aspects of IoT [88]

Therefore, there is a clear need to develop appropriate security mechanisms dedicated for the IoT and find alternatives that aim to balance both normal availability and operability of its services, while maintaining properties of integrity, confidentiality and authenticity, fundamental when developing secure network environments.

With this dissertation, the focus is on the specific case of authentication and fine-grained access control in an IoT environment, presenting a security-oriented architecture solution for managing its facilities. Such solution is then implemented as a prototype using adequate technologies for IoT systems.

1.1 OBJECTIVES

The purpose of this dissertation was to study and develop a working prototype implementation of the proposed theoretical secure architecture model, collectively developed during the Angerona research project at IEETA. This solution was designed with regards to scalability, while abstracting the underlying communication technology that could be employed, thus allowing, in the long run, the execution of multiple IoT applications from different companies or manufacturers without any resulting deployment issues or interference amongst them.

This way, before starting the development of any solution, and in order to fully grasp the IoT concept, it was also implicitly in the scope of this dissertation to investigate about the IoT

nature itself. Such resulting research is presented in the first chapter of the state-of-the-art of this dissertation (Chapter 2) as an entry level overview of the whole IoT domain.

Regarding the security goals of the architecture, one of the main objectives was the implementation of proper access control the access to the IoT devices, i.e, sensors and actuators, and to the data they originate, thus preventing them from being manipulated by attackers. Additionally, one extra concern was regarding the prevention of devices from leaking information to other endpoints other than those properly authorized in a previous authentication process. Therefore, in order to fulfill this aspiration, the access control is enforced in a split approach, through centralized entity controllers and specialized IoT gateways. This authentication process makes use of pre-shared public keys, asymmetric cryptography algorithms and a ticket-based approach for secure session establishment, all without any PKI scheme.

This architecture was then implemented as a physical prototype using appropriate hardware and technologies for the development of IoT systems. BLE was chosen as the underlying communication protocol to connect the IoT devices and Protocol Buffers were applied for performing inter-domain data serialization.

At last, a discussion is presented regarding the security evaluation of the solution, along with some of the identified issues that should be addressed in the eventual future continuation and complementation of this dissertation.

This work was elaborated under the Angerona project, developed at IEETA (Instituto de Engenharia Electrónica e Telemática de Aveiro), which aimed the development of a security-oriented architecture for the deployment of IoT facilities. From the research performed during this project and dissertation, an article entitled "Security-oriented Architecture for Managing IoT Deployments" was written and published in the Symmetry journal [153].

1.2 DISSERTATION OUTLINE

This document is organized in 8 Chapters, the first one, corresponding to the introduction, already presented here. The following chapters of this dissertation are organized in the following sequence:

- **Chapter 2:** introduces essential information about the state of the art of the IoT, including it's usage scenarios, architectures, applied technologies and a brief overview of its overall challenges and issues.
- **Chapter 3:** presents the security related problems affecting the IoT, discussing emerging threats and vulnerabilities in a layered perspective while also covering some current security framework architectures.
- **Chapter 4:** provides an overview of the authentication process in the IoT and the cryptography fundamentals required to understand it. Also presents the authentication protocols in which the developed architecture was based.
- **Chapter 5:** presents the developed solution architecture and its communication protocols, while also describing the security objectives and decision rationale.

- **Chapter 6:** presents the developed prototype of the architecture along with the technologies and frameworks adopted.
- **Chapter 7:** presents an overall security evaluation of the architecture and of the adopted technologies.
- **Chapter 8:** wraps up the dissertation with conclusions and future work, including further investigation, testing and other considerations.

The Internet of Things

The Internet of Things, or IoT for short, is currently a common debated topic and intensively investigated by researchers and developers, mostly due to its increasingly vast number of application areas and use cases. However it faces many obstacles that stop it from reaching its ultimate potential, one of these being at the level of user privacy and security concerns. However, before one can describe these IoT driven problems and their corresponding proposed solutions, it is important to understand the IoT paradigm itself. That means exploring the core concepts and visions of what consists the IoT, how it came to be, different proposed architectures, existing standards, use cases and so on. These topics will be discussed in the current chapter, which describes the current state of the art of the IoT.

2.1 IoT RISE, CONCEPT AND VISION

Ever since its genesis in the 1960s, the Internet has been constantly evolving and adapting to our everyday needs, dramatically revamping our ways of living. It all began with the research commissioned by the federal government of the United States of America in the 1960s, with the objective of a building robust and fault-tolerant communication mechanism with a computer network [126].

From this idea came the Advanced Research Projects Agency Network (ARPANET), an early packet-switching network that later became the first network to implement the protocol suite TCP/IP, which represents the technical foundation of the Internet as we know today, still remaining the most widely used network protocol [19].

Initially used for specific governmental needs, the application of the Internet for commercial purposes quickly became a commonly debated topic and during the late 1980s the first Internet Service Provider (ISP) companies were established. Within the next few years, with an operational and interconnected network infrastructure, conditions were established to serve access to the Internet. However, most of the existing use-cases were still primitive, serving only a small and knowledgeable fraction of the population. Nonetheless, the potential of the Internet was starting to captivate the interest of various companies and businesses, and

in 1989 with the development of the World Wide Web (WWW) by Tim Berners-Lee, the Internet applications and services soon became widely available.

During these evolving stages, however, other Internet related technologies were in development, such as Sensor Networks and NFC using RFID tags, which were also gradually maturing. From their convergence, new visions and possibilities started to emerge, in particular direct Machine to Machine (M2M) communication over the Internet. With this concept in mind, researchers soon realized the benefits of connecting more machines to the Internet with autonomous and self-organizing properties. It was from this vision, that a new connection paradigm was born which would be later referred to as IoT [141].

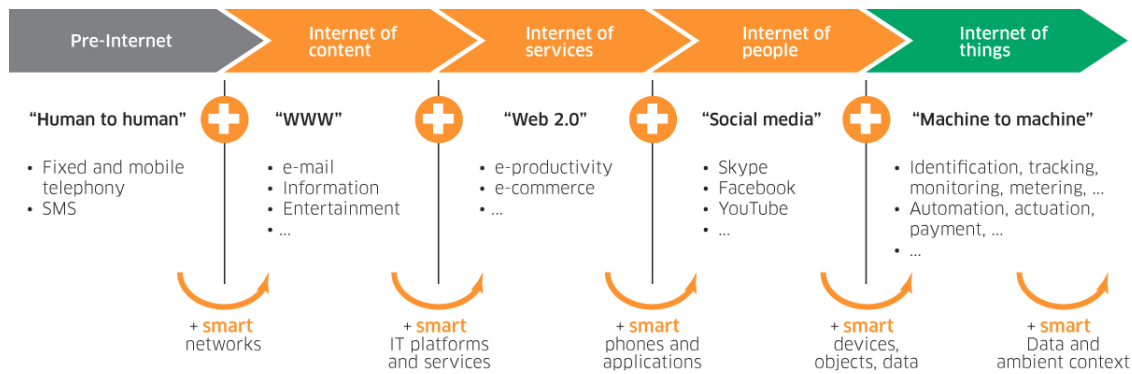


Figure 2.1: Internet evolution towards IoT [73]

The first known IoT appliance is considered to date back to the 1980s, when students of Carnegie Mellon University modified a coke machine by connecting it to the ARPANET. This way they could know how many coke bottles had remained in each one of its rows and for how long. The key element, in order to obtain information about the contents of the coke machine from afar, was to monitor its lights. When one soda bottle was purchased, a red indicator light for the corresponding column would flash for a few seconds before turning back off. However, when a column was empty, the light stayed on until the sodas were replaced. A board was then installed in order to sense the status of each one the indicator lights, and the machine was connected to the ARPANET via a gateway of the department's main computer. By writing a program for the gateway that checked the status of each column light every few seconds, it was then possible to track for how many minutes the bottles had been in the machine after restocking, being registered as "cold" after three hours, while allowing to check if the machine was effectively empty. As such, the students could save the time of a useless trip to the machine, thus avoiding an empty machine or any hot beverage [2], [132]. In the following years, many other appliances containing machines connected to the Internet started to emerge. However, even though the IoT genesis and evolution was set out in the 1980s, the original term was only coined later in 1999 by Kevin Ashton, the Executive Director of Auto-ID Labs in Massachusetts Institute of Technology (MIT) [12].

Nonetheless, it was only in the later years, with the increased availability of the Internet, joined with the cost reduction of electronic devices, that conditions were set for the IoT to flourish. As stated by the author in [104], the reason for the rise of IoT and its commercial

success in the recent years, involves not only those already mentioned conditions, but it was also influenced by other events such as the mobile smartphone revolution, the increasingly deployment and availability of web Application Programming Interfaces (APIs), the growing of the IoT community members, the developments in the programming frameworks and their accessibility and lastly the advancements in the data analytic methodologies.

Either way, the core concepts in the foundation of the IoT are not new or groundbreaking. The technologies of RFID have been used for years in the industrial and manufacturing contexts. For example, an RFID tag embedded to an automobile during its route in the production line helped keeping track of its assembly progress [140]. On the same page, the view of M2M direct communications is also not new as it represents one of the underlying concepts of the Internet where clients, servers and routers communicate with each other.

This way, what is so revolutionary about the IoT is the evolution and spread of these already existing technologies, as they increased, both in number and variety of devices, together with their application and interconnection across the Internet. The traditional Internet initially connected devices, such as servers and desktop computers, followed by laptops and other mobile devices, mainly tablets and smartphones, which were originally designed with integrated processing, storage and network capabilities. However, in the IoT scenario, the objective is to attach sensors and actuators, denominated as “things”, to everyday devices, making them accessible through the Internet, even though they were not initially conceived and designed to operate in this fashion [141].

In the IoT domain, the concept of “things” can be considered any object or person which can be distinguishable in the real world. Everyday objects do not only include electronic devices we encounter and use in a daily basis and other technologically advanced products, such as equipment and gadgets; they also include “things” that are not normally considered to be anything electronic related - food, clothing, furniture, materials, parts and equipment, merchandise and specialized items, landmarks, monuments and works of art and all the miscellany of commerce, culture and sophistication [70]. This means that "things" can not only include objects such as fridges, light bulbs, curtains, or chairs, but also living beings, such as people, animals or plants. This way, "things" can be considered any real objects in the physical world.

Today the IoT vision, that is, a world of fully autonomous Internet connected devices, is rapidly turning into a reality. The exponential growth in number of devices and application areas is blurring industry borders and has led to the rise of many new niche industries, while creating new business opportunities that were previously unknown. According to the data and analytics company GlobalData, the global market for Internet of Things reached 130\$bn in 2018 and is projected to reach 318\$bn by 2023, at a Compound Annual Growth Rate (CAGR) of 20% (see Figure 2.2) [36]. On another study analysis, Gartner predicts that there will be more than 20 billion IoT devices operational by 2020 [37].

There is no denying the enormous potential the IoT holds and how its growth and adoptance will simplify and soothe our lives, creating a wide range of applications using its various adopted technologies. However, IoT developers must be cautious as these systems

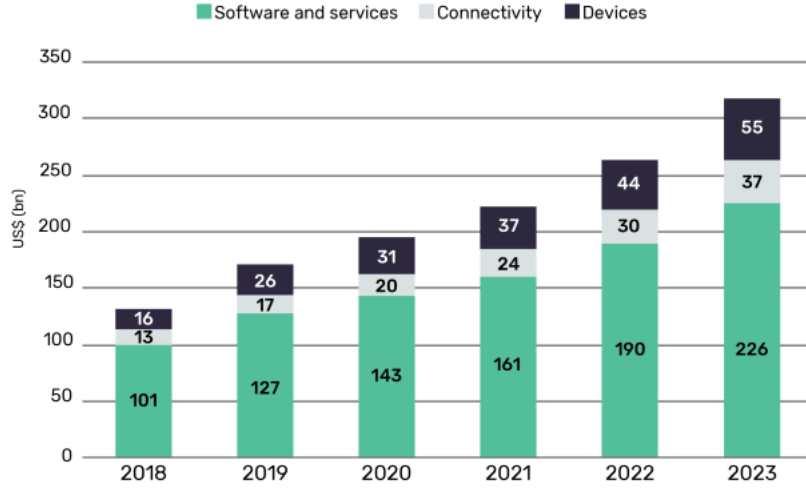


Figure 2.2: Global IoT revenue by technology segment (\$bn) 2018-2023 [36]

will inevitably face problems such as efficiency, scalability and security. Therefore, studying the background and state of the art of IoT is of the most importance when developing any IoT application. Over the next sections of this chapter such essential information will be thoroughly presented.

2.2 DEFINING THE IoT

Defining the concept of what constitutes an IoT, and what distinguishes it from the common Internet, is to this day still not a clear and linear idea. Also, due to the lack of a universal entity to standardize and supervise the IoT, there is no single definition acceptable by the global community of users. As such, several different groups, comprising of academics, researchers, developers, and so on have proposed a definition for this new paradigm. These different definitions and architectural models for the IoT reflect different perspectives and support different business interests. Therefore, it is important to understand and analyze these different approaches when studying the IoT. For this dissertation the IoT definitions from the technological organizations Institute of Electrical and Electronics Engineers (IEEE), Internet Engineering Task Force (IETF), National Institute of Standards and Technology (NIST) and World Wide Web Consortium (W3C) were considered from the many existent, and are as follows [110]:

- **IEEE:** “A network of items each embedded with sensors which are connected to the Internet.”
- **IETF:** “The basic idea is that IoT will connect objects around us (electronic, electrical, non-electrical) to provide seamless communication and contextual services provided by them. Development of RFID tags, sensors, actuators, mobile phones make it possible to materialize IoT which interact and co-operate each other to make the service better and accessible anytime, from anywhere.”

- **NIST:** “Cyber-Physical System (CPS) - sometimes referred to as the Internet of Things (IoT) - involves connecting smart devices and systems in diverse sectors like transportation, energy, manufacturing and healthcare in fundamentally new ways. Smart Cities/Communities are increasingly adopting CPS/IoT technologies to enhance the efficiency and sustainability of their operation and improve the quality of life”
- **W3C:** “The Web of Things is essentially about the role of Web technologies to facilitate the development of applications and services for the Internet of Things, i.e., physical objects and their virtual representation. This includes sensors and actuators, as well as physical objects tagged with a bar code or NFC. Some relevant Web technologies include HyperText Transfer Protocol (HTTP) for accessing RESTful services, and for naming objects as a basis for linked data and rich descriptions, and JavaScript APIs for virtual objects acting as proxies for real-world objects.”

As one can conclude from these definitions, despite their small differences and scopes, they share the common property that physical objects produce some data which is then transported through the standard Internet. As such one may deduce that, while the first version of the Internet was centered around the people and the data they produced with their online interactions, this new paradigm of the IoT will revolve around the “things” and the data they continuously generate.

2.3 WIRELESS SENSOR NETWORK

Most of the IoT devices are small in size and have reduced processing capabilities. They can, however, interconnect, via wireless communication protocols, and form what is known as Wireless Sensor Network (WSN) which allows for wide applications in areas such as environmental monitoring, infrastructure monitoring, traffic monitoring, etc [5]. These devices, referred to as nodes in the WSN scenario, function autonomously and cooperate with the goal of collecting, processing and transmitting the generated data. This data is then delivered across the network through a gateway (also known as sink) that acts as a communication endpoint of the nodes. As such, the WSN concept is one that must be understood as it is one of the foundational paradigms that constitutes the WSN domain. Figure 2.3 depicts an example of a generic WSN network.

Consequently, the WSN features introduce the concept of ubiquitous sensing capability, one of the critical aspects necessary in order to achieve Ubiquitous Computing, the paradigm of providing computational capabilities to all objects in our environment [138]. Thus, when these objects have connectivity to the Internet, however, we enter in the domain of the IoT. As such we can say that Ubiquitous Computing, in the same way as the WSN, is a subdomain of the global context of the IoT, and allow for it to become a reality.

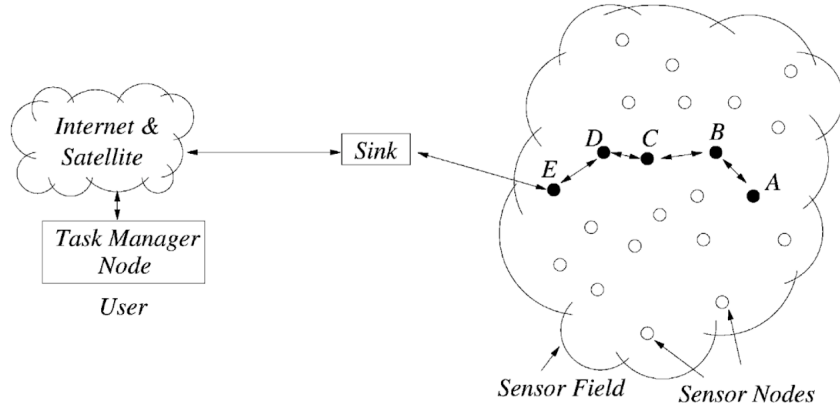


Figure 2.3: Sensor nodes scattered in a sensor field [5]

2.4 CLOUD COMPUTING

One other concept important to understand in order to grasp the full scope of the IoT is Cloud Computing [25]. Since the IoT demands an efficient, secure and scalable computing and storage resourcing, these characteristics can be provided through the implementation of this recent paradigm. It promises reliable services that are delivered through data centers, accessed from anywhere in the world, and applying storage virtualization procedures. As such, the cloud computing data centers act as receivers of data from the ubiquitous sensors, processing, analyzing and interpreting the data, while abstracting these underlying steps from the application user. The Cloud Computing providers may offer their services according to three different delivery models: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) and four different deployment models: Private Cloud, Community Cloud, Public Cloud and Hybrid Cloud [85]. Figure 2.4 depicts a conceptual IoT framework, considering the WSN and Cloud Computing paradigms, including some application areas of the IoT.

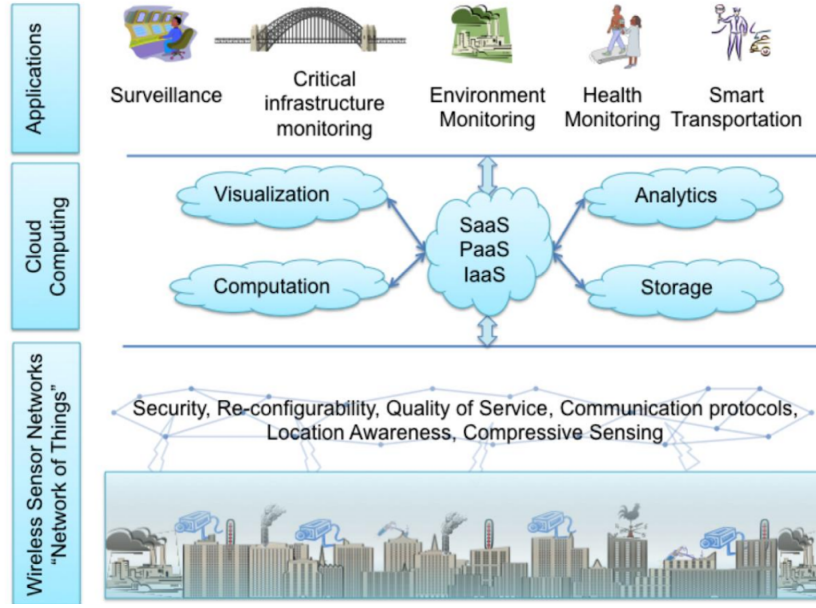


Figure 2.4: Conceptual IoT framework with Cloud Computing at the centre [45]

2.5 APPLICATION AREAS

The domain of application areas in the IoT seems to be unlimited, resulting in new market opportunities for both companies and customers. Its devices are mostly created for consumer use and its application purposes cover areas from connected vehicles to wearable technology, connected health or remote monitoring appliances. As such, they have been thoroughly analyzed and may be classified into four application domains: (1) Personal and Home, (2) Enterprise, (3) Utilities, and (4) Mobile [45].

2.5.1 Personal and Home

In this scenario, the collected sensor information is used only by the individuals who directly own the respective network. For example, the ubiquitous healthcare paradigm, which has been envisioned in the last two decades, can be considered in this category of the IoT since it provides the ultimate platform for the former to exist [14]. For example, using body area sensors and an IoT backend to upload and store data in the servers, it would allow to use a smartphone in order to communicate with several interfaces and analyze the given sensor measurements. Additionally, this could be extended in order to create, for example, a home monitoring system that would aid elder people by allowing doctors to monitor the patients from their homes, which would result in reduction both in hospitalization costs through early interventions and treatments [81].

Similarly, the context of a smart home where all home devices are digitally controlled and managed such as air conditioners, light bulbs, refrigerators, washing machines is another classic case of IoT applications [28]. This one, however, is already happening, as many homes being sold today have some sort of smart devices and control panel. This management may

bring economic benefits such as better energy management and savings. However, this scenario also bears additional security concerns due to the personal character of the data it generates and transports, which is deeply tied with individuals. As such, the underlying IoT platform needs to guarantee the security of such data which, as will be seen in the next chapter, faces many challenges.

2.5.2 Enterprise

IoT applications within a work environment are referred to as enterprise-based applications and the information collected on these networks are generally manipulated and stored only by the company itself. One common application in this category is in the scope of environmental monitoring, which is implemented to, for example, keep track of the number of occupants or manage utilities within a corporate building such as heating, ventilation or lighting. Additionally, sensors have always been implemented in factories for security, automation or climatization control. These could then be replaced or upgraded by wireless systems, allowing more flexibility, resulting in an IoT subnet dedicated to factory maintenance, for example [45].

2.5.3 Utilities

In this application domain, the information from the networks, instead of being for user consumption, they usually aim at service optimization. This scenario is already being implemented by companies for the management of their resources, such as in smart grids and smart metering [147]. By continuously monitoring electricity points within a house and using this information to modify the way electricity is consumed, efficient energy consumption can be achieved. Ultimately this gathered information at a city scale can be used for maintenance and load balancing within the grid, ensuring high quality of service.

One other emerging IoT domain to consider in this category is video-based IoT [4] which integrates scientific areas such as image processing, computer vision and networking frameworks. One common application usage to be considered would be video surveillance. With this enhanced proposed capabilities, offered by the IoT, it would be possible to track targets, identify suspicious activities or monitor unauthorized access on the fly, although sophisticated video analysis is still a recent technology area.

Water network monitoring and quality assurance of drinking water is another example where IoT could be applied by use of sensors measuring critical water parameters installed at strategic locations in order to ensure high supply quality. This network extension to agriculture is another example, which would allow for irrigation and soil monitoring [78].

2.5.4 Mobile

Due to the differences in data sharing and backbone implementation required, smart transportation and logistics are considered in this separate domain. Here, the transport IoT will enable the use of large scale WSNs for online monitoring of travel times, route choice behavior, queue lengths and air pollutant and noise emissions. Also, the IoT is likely to replace the traffic information currently provided by the existing sensor networks of inductive loop vehicle detectors employed at intersections of traffic control systems. Combining this with information

gathered from the urban traffic control system, valid and relevant information on traffic conditions can then be processed and presented, resulting in safety improvements for both pedestrians and drivers [72].

Efficient logistics management is another important mobile IoT application worth considering [77]. This includes the monitoring of items being transported and also the efficient planning of the transportation itself.

2.6 UNDERSTANDING THE IoT ARCHITECTURE

When investigating and developing IoT applications, one important aspect that must be understood is the different existing architectures, either from literature proposed to the ones currently deployed and operational. Although there's a wide number of possible applicational fields in the IoT and there's a diverse number of associated technologies it may incorporate, the core operations performed by a certain IoT application are always the same: the generated data needs to be transported, stored, processed and eventually made available for use in their endpoints, usually clients or data analysts. The IoT devices, that is, sensors and actuators, produce the data and send it to their corresponding service data processing environment, through their corresponding gateway and the Internet itself. For these devices, the first stage of their life cycle corresponds to the acquisition of data and the last being its transmission [117]. They may also process, transform or store the data, however this is usually not the case for most applications, which perform such operations at specialized endpoints (usually web servers) with much higher processing and storage power. The general stages, or lifecycle, of IoT applications are depicted in Figure 2.5.

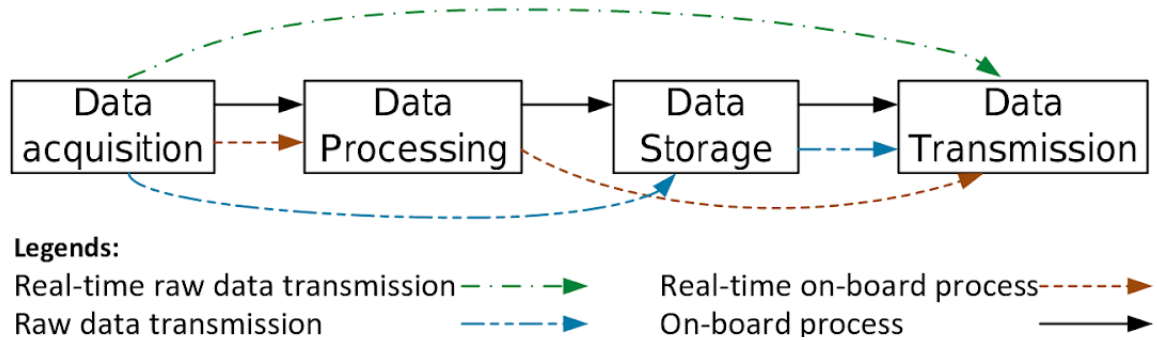


Figure 2.5: General stages of an IoT application [117]

One major problem introduced by the IoT paradigm comes from the very high diversity in both the hardware capabilities and the communication requirements among different types of things. When approaching the problem from a hardware perspective, the things can have different power, memory or communication capabilities. Also, they may have very different Quality of Service (QoS) requirements in terms of delay, energy consumption or reliability. As such, these two characteristics create conflicts when designing a homogeneous framework that takes into account the diverse capability nature of the things. Thus, this motivated the

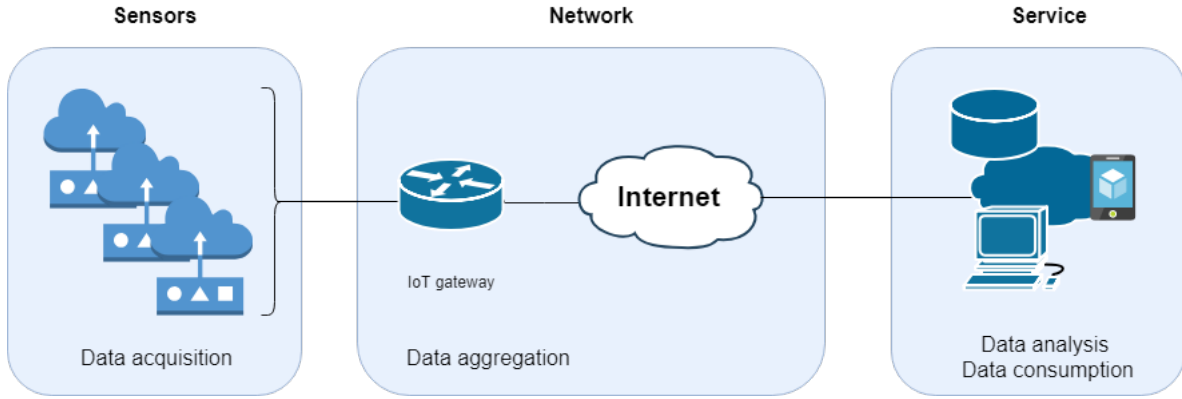


Figure 2.6: Topology of a generic IoT network (adapted from [142])

development of adaptive cross-layer communication schemes in the IoT [48]. A simplified and generic high-level layered architecture can be seen in Figure 2.6. Here, several sensors produce data continuously or based on events in the real world. This data is then sent in real-time to their designated gateway. In case the data is not sent in real-time, it is sent usually based on previously defined constraints or rules. For example instead of generating and sending a temperature measurement every second, the device can operate intermittently every 5 seconds or store this data in memory and only dispatch it after a certain number of measurements or amount of time occurred. This communication to the gateway can occur wirelessly or not. The gateway itself act as a data aggregator, and its function is just to relay the received data into the corresponding service, or, on the other hand, relay data received from the service into the devices. This reverse communication can be due to firmware upgrades, changes in access control policies, authentication operations, etc. At the service layer the data is then collected and processed and will then be used in order to take actions and can be accordingly accessed via service APIs.

2.6.1 Software architecture

In software development, the architecture serves as a blueprint for a system, as it provides an abstraction to manage the system complexity and helps establishing a communication and coordination mechanism among its components. By defining a structured solution to meet all the previously identified technical and operational requirements, it also allows for the optimization of common quality attributes, such as performance and security [137]. Therefore, in a similar fashion with other software paradigms, IoT developers also face the necessity of thoroughly identifying constraints and operational specificities in order to design a proper and robust architecture.

2.6.1.1 Service-oriented Architecture (SOA)

Service-Oriented Architecture, or SOA for short, is an architectural style in software development that aims to achieve loose coupling among the software interacting agents. In this context, an agent is referred to as a “service”, that is, a single unit of work done by a service

provider in order to achieve a desired end result for a service consumer, which is shared across the application domain. As such, SOA manages to achieve a loose coupling by employing two architectural constraints [51]:

1. A small set of simple and ubiquitous interfaces to all participating software agents. Only generic semantics are encoded at the interfaces. The interfaces should be universally available for all providers and consumers.
2. Descriptive messages constrained by an extensible schema delivered through the interfaces. No, or only minimal, system behavior is prescribed by messages. A schema limits the vocabulary and structure of messages. An extensible schema allows new versions of services to be introduced without breaking existing services.

By applying the SOA development principles, it allows for increased business agility, improved business workflows, extensible architecture, scalability, enhanced reuse and a longer life span of applications [71].

2.6.1.2 SOA applied to IoT

As "things" might move or need real-time interaction with their environment, an adaptive architecture is necessary to help devices to interact dynamically with the other "things". Additionally, due to the decentralized and heterogeneous nature of the IoT, it requires that its underlying architecture provides efficient event-driven capabilities. In IoT, the architectural design is concerned with specificities such as architecture style, networking and communication, smart objects, web services and applications, business models and corresponding process, cooperative data processing, security, etc. Therefore, from a technological perspective, the design of an IoT architecture needs to consider extensibility, scalability, modularity, and interoperability among heterogeneous devices. Hence SOA is often considered a good approach to achieve interoperability between heterogeneous devices in several ways. Multiple research areas such as cloud computing, WSNs and vehicular networks, have already implemented SOA successfully, resulting in methodologies that could be applied to the IoT scenario depending on technology, business needs or technical requirements [27]. For instance, the International Telecommunication Union (ITU) recommends that an IoT architecture should consist of four different layers. These layers are: Device layer, Network layer, Service Support and Application Support (SSAS) layer and Application layer [108]. However, some other authors propose similar architectural models with five layers [144], four layers [79], [27] or even three layers [61], [14]. It is important to note that there are many additional proposed architectures that present more differences than just the number of architectural layers. Therefore, for the purpose of this state-of-the-art study, and to avoid stepping out of the scope of this dissertation, only the ones previously mentioned were considered.

For the context of this study, in the next section, the focus will be on the four layer architecture as presented in [27].

2.6.2 The four layer IoT architecture

2.6.2.1 Sensing Layer

In this layer, the smart devices, or "things", with tags or sensors are considered. Their main purpose is to identify objects and to gather information. Thus, this includes 2D barcode labels/readers, RFID tags, Global Positioning System (GPS), sensors and other types of terminal devices. In some industry sectors, Universal Unique Identifier (UUID) are assigned to devices, allowing them to be easily identified and retrieved, making its assignment critical in the deployment of IoT network services [56], [145]. The sensing layer is also often referred to as Perception, Physical or Device layer in other proposed architecture schemes due to their core similarities.

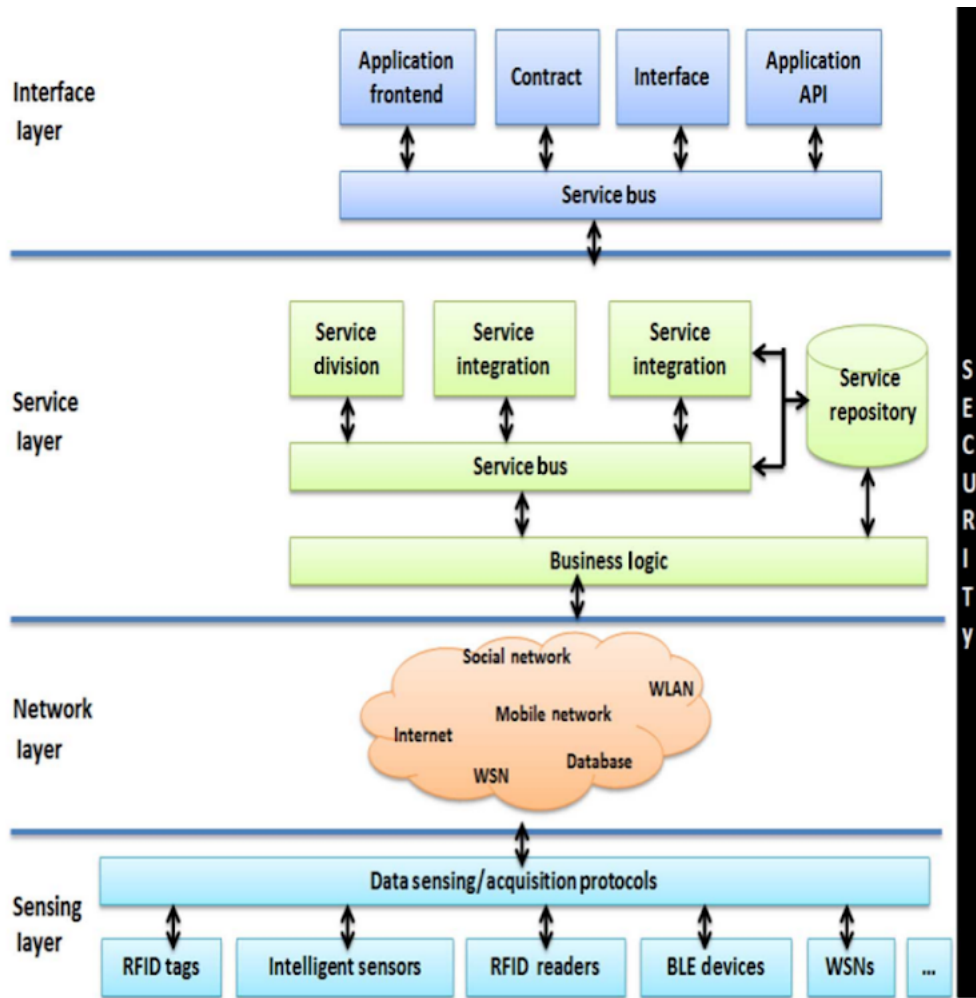


Figure 2.7: Functionality perspective of SOA four layer IoT architecture [27]

2.6.2.2 Network Layer

The purpose of this layer is to interconnect the IoT devices, allowing them to transmit the information collected at the sensing layer through the common Internet. In the SOA-based

IoT, the provided services are generally deployed in a heterogeneous network and all related "things" are brought online into the service Internet [48].

The design of the networking layer needs to address issues such as network management technologies for heterogeneous networks (such as fixed, wireless, mobile, etc), energy efficiency, QoS requirements, service discovery/retrieval, signal and data processing and privacy and security concerns [46].

This layer is also often referred to as the Transport layer.

2.6.2.3 Service Layer

The service layer is built upon middleware technology that provides functionalities that ease the integration of services and applications in the IoT environment. This middleware facilitates the development of a cost-efficient platform, allowing the reuse of both software and hardware. As such, this layer should be able to identify common application requirements and provide APIs and protocols to support necessary services, applications and user constraints. It also processes all of the service-oriented derived problems, including information exchange and storage, data management, search engines and communication [27]. In essence it is responsible for the creation and management of the IoT services, while making them available to end users.

2.6.2.4 Interface Layer

The interface layer aims to simplify the management and interconnection of the IoT devices, providing interaction methods to users and other applications. Due to the large number of operational IoT devices, made by different vendors and manufacturers, it results in the coexistence of different standards/protocols, which could consequently lead to problems regarding information exchange, communication between the things and cooperative event processing among different devices. In order to ease the management and interconnection of the devices, the interface layer may apply what is known as an Interface Profile (IFP), that is, a subset of service standards that support interaction with applications deployed on the network [27].

2.6.3 Hardware architecture

A generic hardware architecture of an IoT System on a chip (SoC) device is present in Figure 2.9. An IoT embedded device has many –if not most– of these components, e.g. at least one RF (Radio Frequency) component for the connectivity [117]. For the purpose of this dissertation only the Radio Frequency (RF) technologies will be considered and studied (see Section 2.7).

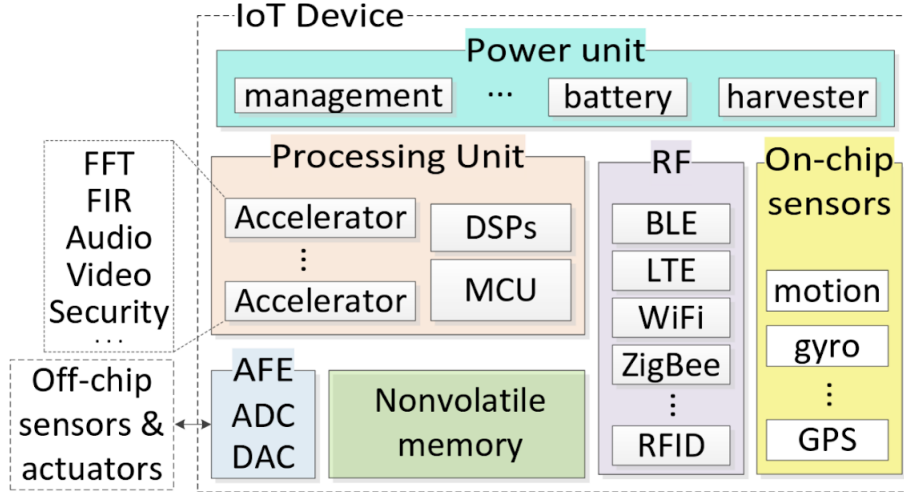


Figure 2.8: General architecture of an IoT embedded device [117]

2.7 IoT ENABLING TECHNOLOGIES

Due to its heterogeneous nature, multiple enabling technologies can be used in the development of an IoT environment. In this section the most relevant ones for this dissertation will be considered and briefly described.

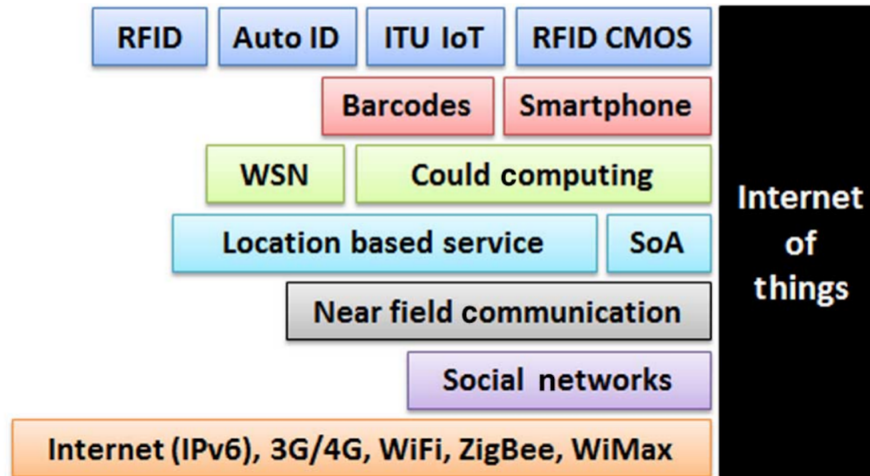


Figure 2.9: Technologies associated with IoT [27]

2.7.1 Radio Frequency Identification (RFID)

Radio Frequency Identification RFID is a contactless and automatic identification technology which enables smart-labels to identify and track tags attached to objects. This process is done without requiring manual intervention and can operate in a variety of harsh environments.

An RFID system may consist of several components such as tags transponders, tag readers, antenna and interface. Based on the method of power supply provisioning, it can be classified

into three categories: Active RFID, Passive RFID and Semi Passive RFID [128].

RFID is considered to be in the foundation and networking core of the construction of the Internet of Things and plays an important role in this domain for solving identification issues of the things in a cost effective manner [76]. RFID has an extensive range of wireless applications such as distribution, tracing, patient monitoring, military apps, etc [90].

2.7.2 Near Field Communication (NFC)

Evolving from RFID, Near Field Communication (NFC) is a set of short-range wireless technologies that allows for two devices, usually mobile devices, such as smartphones, to exchange data by bringing them close with each other (usually 4 cms) [34]. This communication involves two entities: an initiator and a target. The initiator actively generates an RF field capable of powering the passive target device. Once it is activated, small amounts of data are transferred between the two. The target device usually takes simple physical forms such as tags, stickers or cards since they run on very low amounts of power, being more power-efficient than other wireless methods. This way, NFC is considered an enabling technology for the IoT as it may bring all the unpowered objects that lack internet connection to be online and a part of this new Internet domain [54].

2.7.3 Wi-Fi

Wi-Fi is an IEEE 802.11 standard that uses radio waves to provide network connectivity within a 100m range, being one of the most popular wireless communication technologies. A Wi-Fi connection is done through the use of a wireless adapter which creates a hotspot, that is, an area in the range of a wireless router which is connected to the network and is responsible for granting its users Internet connectivity. Based on the amount of data on the network, Wi-Fi may transmit in the frequencies between 2.4GHz and 5GHz [106].

With the upcoming age of the IoT, it would seem feasible to take advantage of the existing and vastly deployed Wi-Fi infrastructure to connect the new devices. However, Wi-Fi was originally developed for high bandwidth applications, and as such the classic Wi-Fi may not be feasible for low-power IoT environments, especially for devices operating on batteries. However, the IEEE 802.11 standard defines a power saving mechanism which allows the mobile devices to operate in a power saving state where they may turn off the receiver and the transmitter, resulting in a viable alternative for the IoT [134], [135]

Additionally, Wi-Fi has the capability of creating wireless ad-hoc networks (ad-hoc mode), that is, decentralized networks that do not rely on a router for devices to communicate with each other. This Wi-Fi feature is also considered suitable for the IoT as a result of the increased necessity for eliminating system complexity and fixed monolithic infrastructures, such as in the industrial IoT scenarios [60].

2.7.4 Bluetooth

Bluetooth, or IEEE 802.15.1 standard, is a wireless communication technology developed for short-range and low-end devices with the intent of replacing cables on computer peripherals. It is a packet-based protocol and its architecture is of master/slave type defining two connectivity

topologies known as piconet and scatternet. The piconet is a Wide Personal Area Network (WPAN) that is formed by a Bluetooth device, acting as the master in the network and is defined by a frequency-hopping channel based on the master address. The remaining devices operate as slaves and are synchronized using the clock of the master. The scatternet represents a collection of operational Bluetooth piconets that overlap both in time and in space, where devices may participate in several piconets at the same time, allowing information to flow between them [75], [103].

In order for Bluetooth to adapt and be applicable in the upcoming IoT reality, it would need to reduce its power consumption so that it could be used in battery powered and resource constrained devices. Additionally, the IoT demands both the reduction of the transmitted data and the communications between devices. These properties diverge from the core Bluetooth appliances where the devices constantly interact with each other and usually transfer larger amounts of data. From this realization, in 2010 the Bluetooth Special Interest Group (SIG) introduced BLE, first specified in Bluetooth 4.0, also known as Bluetooth Smart [26].

2.7.4.1 *Bluetooth Low Energy (BLE)*

Bluetooth Low Energy was designed with the objective to tackle the IoT communication requirements by introducing a new radio and new interfaces from its predecessor, known as Bluetooth Classic, allowing two devices to quickly connect with each other [26].

The channel bandwidth in BLE has also doubled to 2MHz resulting in a total of 40 channels, of which three are used as advertising channels and the remaining as data channels. The purpose of the advertising channels is to ease the discovery of BLE devices and the establishment of the initial communication between devices. The data channels are used for all of the data communications. BLE works based on four different protocols from the BLE protocol stack [111]:

- Logical Link Control and Adaptation Protocol (L2CAP): responsible for the packet fragmentation and posterior reassembly.
- Generic Access Profile (GAP): defines procedures for BLE device discovery and managing the connections.
- Attribute Protocol (ATT): defines how the server exposes its data to a client and how it is structured.
- Generic Access Profile (GATT): describes a framework for device discovery and operate on characteristics of a BLE peer device.

These protocols will be further explored in chapter 6 (see Section 6.1.4) as they are required to understand how the BLE communication and stack come into play in a practical perspective.

2.7.5 **ZigBee**

ZigBee is a radio frequency communication standard, based on IEEE 802.15.4, designed to address the unique requirements of sensors and control devices: low latency, low data rates and very low energy consumption [66]. As such, these properties make ZigBee a suitable communication mechanism to be employed in IoT systems [40]. The ZigBee network supports

star, tree and mesh network topologies and a device in a Zigbee network may be of three types [32]:

- ZigBee Coordinator (ZC): Precisely one coordinator is present in every network and is responsible for its bootstrap. It also acts as a trust center and as a repository for security keys, storing information about the network.
- ZigBee Router (ZR): Acts as an intermediate router that permits data to pass to and from it to other devices of the network.
- ZigBee End Device (ZER): Only communicates with their parent node, a router or the coordinator, while having limited functionality and not relaying data from other devices.

ZigBee also employs security mechanisms built upon the basic security frameworks of IEEE 802.15.4, allowing secure communications, the protection in the establishment of cryptographic keys, and the confidentiality, integrity and authenticity of the transmitted frames [66].

2.7.6 DTLS

Datagram Transport Layer Security (DTLS) [109] is a communication protocol designed to provide privacy and protection against message tampering, forgery and eavesdropping to datagram protocols. It is based on the Transport Layer Security (TLS) protocol, providing equivalent security guarantees while preserving the datagram semantics of the underlying transport. However, the main difference from its predecessor is that DTLS was designed to work under unreliable communications using User Datagram Protocol (UDP) while TLS runs under Transmission Control Protocol (TCP). This property makes DTLS an appropriate protocol to be used in order to secure IoT networks where the communications must be as lightweight as possible. DTLS is mostly applied alongside the application protocol CoAP which will be presented next.

2.7.7 Constrained Application Protocol (CoAP)

The Constrained Application Protocol (CoAP) [120] is a transfer protocol from the IETF Constrained RESTful Environments (CoRE) Working Group. It is specialized for constrained networks and nodes, designed with a focus on M2M. Since M2M communications are heavily present in the IoT domain, CoAP is considered a good alternative over HTTP. It is a request/response based model with built-in service and resource discovery, multicast support, and asynchronous message exchanges. It was developed to easily interoperate with HTTP by adopting a Representational State Transfer (REST) architectural style and Web Universal Resource Identifiers (URIs). Since it runs over UDP, the communication between clients and servers is done through connectionless datagrams with less reliability. However, it provides two different levels of QoS by introducing “confirmable” and “non-confirmable” messages. The former must be acknowledged by the receiver with an ACK packet while the latter require no type of confirmation [93].

2.7.8 Message Queue Telemetry Transport (MQTT)

Introduced in 1999, Message Queue Telemetry Transport (MQTT) is a simple and lightweight publish/subscribe message protocol designed for both constrained devices and unreliable

networks, with low-bandwidth and high-latency. Its design objectives were to ensure reliability and guaranteeing message delivery while minimizing the network bandwidth and device resource requirements [91]. Due to these properties, MQTT is considered a suitable protocol for M2M communications, thus applied in mobile and IoT applications [93]. Being a publish/subscribe message model, it requires a mediating entity also known as message broker. When a client needs to send data it publishes messages to an address, also known as topic, at the broker. In order to receive that message, other clients subscribe to the same topic and the broker will ensure its delivery. The communication between client and broker is connection-oriented, as it uses TCP for its transport protocol, and it ensures security by TLS and Secure Socket Layer (SSL). It also provides three levels of QoS, allowing more flexibility for the underlying communication.

2.8 IoT PROBLEMS AND CHALLENGES

The IoT technologies and applications are still in their childhood, thus facing many research challenges and open issues. This way, it is essential to understand the IoT characteristics and requirements on factors such as cost, security, privacy and risk, before IoT can be widely accepted and deployed, as well as identify the technological barriers stopping this paradigm from fulfilling its ultimate objective: a world where every single object operates autonomously while being connected to the Internet [27]. In this section concerns of standardization, addressing and networking issues, security and privacy will be considered and discussed. Table 2.1, adapted from [14], presents a brief overview of the considered open research issues in the IoT domain.

2.8.1 Standardization

One major issue resultant of the rapid growth of IoT is that it hinders its standardization. Hence the necessity of a standard is of greater value for the further development and wide adoptance of the IoT. It allows for new service providers to develop their applications with ease, lowering the associated entry barriers, resulting in better overall performance of the system. However the design of a global standard demands high coordination efforts in order to ensure devices and applications from different manufacturers, developers and countries are able to exchange data and coexist in the same environment [89]. These standards should also be willing to embrace emerging technologies and should include industry-specific guidelines in order to implement IoT in any industrial environment and facilitate the integration of its services. Some of the recurring issues present in the IoT standardization include interoperability, radio access level, semantic interoperability and security and privacy [27].

Open issues	Brief description
Standards	There is no integrated framework, despite several standardization efforts.
Mobility support	While there are several proposals for object addressing, these do not support the mobility scenario of the IoT, where scalability and adaptability to its heterogeneous technologies are important issues that need to be solved.
Naming	Object Naming Services (ONS) are needed to map references between object description and associated identifier, and vice-versa.
Transport protocol	Current transport protocols are not suitable for the IoT due to their connection setup and congestion control mechanisms. Furthermore, they require excessive buffering to be implemented in objects.
Traffic characterization and Qos support	Data traffic originated by the IoT will produce different patterns than those of the current Internet. As such, appropriate QoS requirements and support schemes are needed.
Authentication	Authentication is a challenge in the IoT scenario since authentication infrastructures currently used are not available. The constrained computational resources of the devices also pose new concerns. Man-in-the-middle attacks are also a great problem.
Data integrity	Data integrity is usually ensured by data protection using passwords, however IoT technologies do not support for big length passwords, being unable to provide strong levels of protection.
Privacy	Private information about individuals may be collected without their awareness. With the current techniques it is impossible to control the diffusion of such information.
Digital forgetting	The collected information about individuals may be retained indefinitely mainly due to the decrease of storage cost. Data mining techniques may be applied to easily retrieve any information even after several years.

Table 2.1: Open IoT research issues (adapted from [14])

2.8.2 Addressing and networking

Since the IoT will employ billions of connected nodes and terminal devices, these must be able to be individually identified and accessed, independently of geographical location or network environment they may be deployed in. This requirement is solved by the use of addresses and addressing policies. As such, an address must employ properties of uniqueness, reliability, persistence and scalability [45].

The original protocol that solved the addressing issue, still in use today, is the Internet Protocol version 4 (IPv4), as it is still responsible for routing most of the Internet traffic. This protocol, however, since the early Internet days, was predicted to fall short as the number of available addresses would not be capable to serve the exponential growth of the Internet

connected devices. Its successor and most recent version, Internet Protocol version 6 (IPv6), is currently being adopted as it allows for an astonishing pool of 2^{128} unique addresses, which are unlikely to be depleted any time soon. In addition, IPv6 provides other benefits and improvements over IPv4, such as hierarchical address allocation methods that facilitate route aggregation, thus limiting the expansion of routing tables and increasing routing performance.

In the scenario of IoT addressing, one introduced major problem is splitting between location and identification of a device. Although the adoptance of IPv6 for the IoT seems like a feasible option to solve the individuality problem of IPv4, the heterogeneous nature of the wireless nodes, variable data types, concurrent operations and confluence of data from devices, make it so that it does not solve the problem in its entirety [152].

Therefore, suitable solutions must be developed in order to solve this problem. These solutions must be able to interrelate heterogeneous schemes that consider diverse location/ID splitting techniques across different domains. Some of considered addressing and networking alternative schemes for the IoT currently being researched and developed are:

- IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN) [74]
- Uniform Resource Name (URN) [115]
- Object Naming Services (ONS) [33]
- Ubiquitous ID (uID) [116]
- Named Data Networking (NDN) [148]

2.8.3 Security and privacy

The growth of the IoT gave birth to new security issues, while aggravating some of those already existing in other Internet connected appliances, making it extremely vulnerable to attacks. The considered main reasons that led to this event are due to the heterogeneity and the large scale of the connected objects. Since its components are operating continuously and are left unattended for the most part, this results in them being easily accessible, conditions that eventually lead to them being physically accessed and attacked. Additionally, most of the IoT derived communications are wireless, which facilitates eavesdropping attempts. One last condition that further aggravates the security IoT problem is that its components have low computing resources (processing power, memory size and storage capability) and low energy capabilities. Due to these constraints it is not feasible to implement many of the complex security schemes already in use by other web services [14], [150].

As such, it is of major importance to develop adequate security and privacy schemes oriented for the IoT ecosystem in order to develop secure IoT applications that will not put in danger the privacy of its users and the Internet as a whole. The vision of a world of fully connected smart objects holds enormous promise both for the businesses and for our everyday lives, but in order for it to succeed the technical challenges and the severe threats that it originates must be thoroughly researched and overcome. Over the next chapter the security and privacy issues will be thoroughly presented and discussed in more detail and some of the current considered solutions will be presented.

IoT Security overview

With the increasing development and sales by industry and startup companies of devices and services connected to the Internet, that is, in the IoT domain, the security and privacy concerns will inevitably increase alongside it. Traditionally most of these companies never had to worry about any type of security whatsoever but this emerging frenzy of being marketable and competitive by turning all devices “smart” is resulting in an increasing number of devices operating with unacceptable and outdated security methodologies or, in the worst case scenario, none whatsoever [18]. As these companies prioritize customer convenience, functionality, compatibility requirements and time-to-market, securing the IoT turns into a real challenge that cannot be overlooked.

As stated in a 2014 study by HP [52], 70% of the most popular IoT devices contain serious security issues, containing on average 25 vulnerabilities, while Symantec reported a 600% increase in attacks against IoT devices from 2016 to 2017 [131]. This reality poses as a great concern for developers, companies, researchers and individuals while creating a great array of opportunities for hackers to exploit, resulting in potential damages to the economy, businesses, corporations and to the privacy and safety of individuals.

Therefore, both security and privacy must not be considered as just an add-on for devices and applications which can be handled in later stages of development, but rather be an integral part of the IoT devices, reliable functioning throughout its lifecycle. This includes ensuring proper patching deployment on devices as soon as any vulnerability is detected on the system and developing security infrastructures with new systems and protocols capable of reducing possible threats related to scalability, availability and security of the IoT ecosystem [92].

Additionally, since the IoT paradigm is fundamentally composed by a collection of interconnected networks and heterogeneous devices, it unavoidably inherits the conventional security issues related to computer networks. Plus, the limited nature in computing, storage and network capabilities of the IoT devices further aggravates the security problem, turning them more vulnerable to attacks than the traditional smartphone or tablet [65].

One other property that greatly distinguishes the security vulnerabilities in the IoT is

that its devices produce and collect personal and sensitive information regarding their users, in some cases without them being aware. As such in the IoT ecosystem, the data becomes a lot more intimate than in the traditional Internet services. Also, due to the large scale of the IoT, massive amounts of data will be generated and collected which will unavoidably travel through the network. This property increases the chances of any leaks or breaches that could have potential harm for the targeted individual, as a malicious party could extract the desired information and compromise the individuals privacy [149]. Furthermore, it also allows for information about the individuals to be inferred by having access to information that originally did not pose any danger or revealed anything sensitive by itself. For instance by accessing the data produced by temperature sensor inside a house, information which one would not considered to be sensitive, it would be possible for an outside agent to infer which periods of time the temperature would be higher due to the inside heating being on, thus concluding that there were people in the house, resulting in the monitoring of the individuals presence in their homes.

The security in IoT is unique due to it not being limited to the assurance of principles of confidentiality, integrity, non-repudiation, but must also consider the physical world aspects, that is, the devices hardware and physical world interactions it creates. As such, the compromise in these devices originate new security risks since it may lead to the physical harm of property and individuals, concerns that were previously not considered in the cybersecurity domain [18].

3.1 IoT SECURITY REQUIREMENTS

In the traditional Information technology (IT) and network systems, the security requirements/objectives generally prioritized are Confidentiality, followed by Integrity and Availability. However in the IoT scenario, due to its large scale and heterogeneity of devices, its systems cross multiple domains and allow for a wide range of use cases, resulting in increased complexities and different levels of necessary security mechanisms. As such, these objectives are directly influenced by the application or service being considered and must be designed in such fashion [57]. In the IoT security literature the proposed security requirements and goals have different scopes and granularity. However they all agree that the security triad or CIA triad [113], which is a model designed to guide policies for information security and used in security development, corresponds to some of the desired goals for the IoT field [92]:

- **Confidentiality:** is the ability to provide confidence to a user about the privacy of any sensitive information in such a way that it prevents an unauthorized entity from accessing it. It can be achieved using different security mechanisms, such as data encryption and strong access control, namely biometric and two-step authentication. In the IoT scenario this property assures that the nodes in the IoT network do not reveal any information to any bystander nodes and also that, in the radio frequency transmission, the tags do not transmit any data to unauthorized readers.
- **Integrity:** is the property that guarantees the protection of information from external interference during its transmission and reception, stopping it from being tampered

without it being known by the system entities. The most adequate methods for ensuring this property are cryptographic hash functions and digital signatures. Version control mechanisms are also often considered for assuring integrity, however it mostly refers to scenarios where data could be changed or deleted by authorized users by mistake, thus not particularly important regarding the properties here presented.

- **Availability:** ensures the reliable access and use of an authorized entity to the data even when under abnormal circumstances. This property also considers prevention mechanisms against bottleneck occurrences and Denial-of-Service (DoS) attacks. These methods are, for example, the configuration of firewalls in the network, eliminating single points of failure by employing redundancy and the implementation of automatic failover techniques.



Figure 3.1: The CIA triad [107]

However these three principles alone are not enough to provide strong security in this new paradigm shift that is the IoT. As such, as researchers in the IoT security topic present, there are more requirements that must be met. In [92] the authors consider the primary security requirements for the IoT to be the insurance of adequate identity authentication mechanisms and to guarantee confidentiality in the exchanged or stored data. In [65] the authors present a taxonomy of security issues and consider the following four security requirements that must be established for a secure IoT deployment:

- Data privacy, confidentiality and integrity
- Authentication, authorization and accounting
- Availability of services
- Energy efficiency

In [44] GSM Association (GSMA) presents a document establishing security guidelines and considers the security challenges inherent to the IoT growth to be Availability, Identity, Privacy and Security. By Identity it is considered the authentication of endpoints, services and customers or end-user operating an endpoint, Privacy the reduction of potential harm to users and Security the integrity assurance of the system.

In [119] the considered security goals are the classic CIA but also including Authentication, Authorization and Accountability. Authentication and authorization are introduced due to wireless communication nature of the IoT systems. Accountability is also stated to be necessary in order to improve the robustness of the IoT services.

Therefore, from the study of different IoT security related publications, the considered security requirements that must be assured in the development of a secure IoT environment are:

- **Confidentiality**
- **Integrity**
- **Availability**
- **Non-repudiation:** ensuring that it is possible to demonstrate that a certain action, performed by an individual or the system, cannot be denied later on, being impossible to deny the authenticity of its occurrence.
- **Authentication:** confirming and ensuring the identity of the source of data in a system. In the IoT this means that services and peers can guarantee to whom and what they are communicating, being able to authenticate and identify other objects in the network while having proof of their own identity, usually by a set of security credentials.
- **Authorization:** is the attribution of rights and privileges to a certain entity or resource, usually in the form of access policies, making it able to access information that was previously inaccessible.

However these are general security practices. In the end, it is the type of system in question, the operational devices and the data being transmitted that ultimately establishes the security rules that must be implemented. Nonetheless, in the real world some devices may not be able to satisfy every intended requirement due to their low-cost nature. Therefore security designers must find a balance and consider the trade-offs between constraints and risks, while documenting the operational implications of a device in case of a security downgrade and must define contingency plans in case of abnormal occurrences [59].

Additionally the security requirements must not be static during the lifecycle of the system. For a secure environment to succeed developers must be ready to fix any occurring new vulnerability or threat. This includes supervising of the system and the deployment of security patches as soon as the anomaly takes place. The IoT Security Foundation's Best Practice Guides [59] provides security guidelines for developers looking to build their IoT products or systems. It covers classification of data, physical security, device secure boot, secure operating system, application security, credential management, encryption, network connections, securing software updates, logging and software update policy.

3.2 IOT SECURITY THREATS / VULNERABILITIES

In order to evaluate the security of a digital system and to protect it from attacks, the threats and vulnerabilities affecting the system must first be identified and studied. Only this way developers are able to apply correct and safe security measures, even though it is impossible to predict all of the outcomes and ways that the system may be exploited [102]. This is especially

important in the IoT since, due to the devices resource constraints, intrusion detection systems and antivirus are infeasible and cannot be adopted to secure the IoT environment.

In this section some of the literature considered security threats and vulnerabilities affecting the IoT are presented.

In [146] the authors consider two IoT devices and develop a case study on their vulnerabilities from hardware, software, and network levels. At the hardware or physical level, upon analyzing the device processor datasheet, they were able to locate its Universal Asynchronous Receiver-Transmitter (UART), which then led to being able to read serial data and eventually change the boot parameters and extract login information. From a network perspective, a network scan was performed and it showed that the device had a running telnet server. Since the shell was running in root mode, the hashed password is extracted and later revealed through brute force, in this case performed by hashing every combination of characters and checking if it is the root password of the device.

In [125] the authors analyzed and tested IoT devices and presented 6 common vulnerabilities and exposures found among them. These are:

- Cleartext local API - Unencrypted local communications
- Cleartext Cloud API - Unencrypted remote communications
- Unencrypted storage - Collected data stored on the disk is not encrypted
- Remote shell access - Command-line interface available on a network port
- Backdoor accounts - Local accounts have default or easily guessed passwords
- UART access - Device is compromised by physical local access

In [119], seven crucial attacks are listed that must be thoroughly addressed when considering the secure development of an IoT system. These are:

- **Physical attacks:** These attacks occur by physically accessing the IoT device, many of which are widely deployed in outdoor environments, thus facilitating its occurrence. This way attackers are able to extract confidential information, modify its firmware, deploy a backdoor or just kill the device in operation. Also in this section, side channel attacks are considered, in which timing information, electromagnetic leaks, power consumption and sound of the device provide data that could be exploited. Researchers have already proven it is possible to extract Rivest–Shamir–Adleman (RSA) and Elliptic-Curve Cryptography (ECC) cryptography keys using this attack method [38], [39].
- **Eavesdropping:** Refers to the process of listening to an ongoing communication, without the involved parties being aware of it. In the IoT, since transmissions are mostly done through wireless environments, which are open access, makes this a very common occurrence and is in the origin of the next two attacks.
- **Impersonation:** Occurs when a malicious entity tries to pose as another legitimate one. A very common attack derived from this process is known as a MITM.
- **Man-In-The-Middle (MITM):** In this attack, the perpetrator relays or alters the messages transmitted between two communicating parties, which believe they are communicating with each other. This attack is very useful in order to break the confidentiality and integrity when using public-key cryptography. By replacing exchanged

keys with his owns, a secure channel is established in which the attacker has access to the exchanged message contents, rendering the cryptography protection useless.

- **Denial-of-Service (DoS):** This type of attack tackles the availability of the whole IoT system by flooding the target resources connections with a huge number of requests, thus overloading the network, preventing users from fulfilling their legitimate requests. In the same fashion, denial of service could be performed by triggering any heavyweight operation, consuming computational, memory, network or energy resources, which are critical for the resource constrained devices of the IoT.
- **Access attacks:** This type of attacks involve any kind of attack that results in a malicious entity gaining remote access to the IoT system or its devices.
- **Other attacks:** Firmware attacks, Random-Access Memory (RAM) attacks, ransomware, etc.

From a standardization perspective, the security report for the oneM2M standard [99] provides an extended listing of security issues that could occur based on their specific use cases and map these issues to security requirements, while deriving possible counter mechanisms.

The Open Web Application Security Project (OWASP) [101] presents the top 10 vulnerabilities and bad security practices occurring in 2018 affecting the IoT domain. These are as follows:

1. Weak, guessable or hardcoded passwords
2. Insecure network services
3. Insecure ecosystem interfaces
4. Lack of secure update mechanisms
5. Use of insecure or outdated components
6. Insufficient privacy protection
7. Insecure data transfer and management
8. Lack of device management
9. Insecure default settings
10. Lack of physical hardening

3.3 ARCHITECTURAL LAYERS SECURITY

As depicted in Section 2.6.2 , the architectural models in IoT are approached in a layered fashion usually composed by three major layers: Perception layer, Network Layer and Application Layer. As such, the security problems and measures may also be categorized as belonging to each of these layers [92], [114], [130], [151]. Since the security related problems are complex and difficult to grasp, when designing an IoT environment, to better define countermeasures strategies it is important to understand the different concerns on each layer and the potential attacks that could occur.

3.3.1 Perception layer security

This layer is at the bottom of an IoT architecture and is responsible for collecting and generating information through physical equipment, usually sensors. One of the major issues occurring at this layer is that the perceptual nodes, which have high variety and heterogeneity, are usually constrained making it infeasible to apply frequency hopping communication and protection by use of public key encryption algorithms [92]. As such, this layer is vulnerable to a wide range of attacks that could affect both the devices and the data they collect [43].

Here the privacy is also of major concern since the sensors will be all around us silently collecting data, some of intimate character. As such mechanisms and regulations for human and object privacy protection are deemed necessary. An attacker could capture or inject malicious code into an existing node or even add a rogue node to the network. In the worst case he could even capture a gateway and thus getting control over the entire network. Additionally, the collected data is mainly transmitted via wireless and as such these signals are exposed to the public. If there are no proper protection measures, they could be monitored, intercepted or tampered. This way, they are affected by attacks such as man-in-the-middle attack, timing attack and DoS attack. A replay attack could also be used to affect the authentication process of the physical devices. Since the sensors are operating autonomously without supervision, attackers could physically access them with ease, controlling or even destroying them. A Differential Power Analysis attack is an example of a physical attack that can be very effective in this scenario [67].

In order to secure the perception layer, authentication and access control schemes are a must in order to identify network nodes and their privileges. Data integrity and confidentiality schemes are also important so that the collected data is protected against exposure and tampering. However, as stated in [83] the confidentiality of the sensor data can be considered of low necessity since, in some scenarios, an attacker can place his own sensor and sense the same original values. Obviously this is deeply tied with the scenario in question, since confidentiality must still exist when dealing with more sensitive data, such as of personal character. Authentication is necessary for the prevention of inclusion of malicious nodes in the network, as well as the illegal access to legitimate nodes and their produced data. Data encryption is also needed as it is the mechanism through which the confidentiality of the transmitted data is guaranteed. A key agreement scheme is also here considered and of major importance as the key will be used for authenticating network objects, encrypting data and signing messages. However, the stronger the security measures above mentioned, the higher resource consumption they will require. Lightweight cryptography is a research area that aims to solve this issue, especially regarding the IoT domain. Mechanisms of integrity and authenticity of sensor data is also an area currently being researched [130]. Intrusion detection systems can be used to detect malicious nodes and protect the whole network against insider attacks.

In [151] the security measures from an RFID and WSN perspective are presented, as they are an integral part of the perception layer. As RFID security measures are considered: access control, data encryption, Internet Protocol Security (IPsec) secure channel, cryptography

technology and physical security schemes. As WSN security measures are considered: key management, cryptography algorithms (symmetric key cryptography and public key cryptography), security routing protocols, intrusion detection technology, authentication and access control and physical security design.

3.3.2 Network layer security

This layer is responsible for the transport of the data gathered in the perceptual layer to its destination across the network with reliability. It can be considered of two types: wired or wireless. The latter is the one where the majority of security issues arise since it is exposed to various types of attacks due to the openness of the wireless channels, which could be easily monitored. Thus, despite Internet security mechanisms being mature and applied in many different scenarios in the traditional Internet, they are still target of many attacks. The most common one, the DoS, can occur in the IoT when a large number of malicious nodes overloads the networks with more data than they can possibly handle. With the rise in number of devices in the IoT network, this type of attack is only expected to be more prevalent and severe. Nonetheless, even in a relatively considered safe environment, man-in-the-middle attacks, eavesdropping and replay attacks may still occur. Other problems occurring in this layer are in authentication establishment, data integrity and confidentiality, privacy disclosure and routing security problems.

In the actual IoT structure, the network layer is based on the traditional Internet as a communication network. This way all of the issues affecting it unavoidable also damage the IoT. Plus, in IoT, node arrangement may have randomness associated to it, as well as the already mentioned unreliable communication and energy properties and autonomic behavior. As such, the network layer is ultimately affected due to the dynamic topology and lack of established infrastructures, being an easy target for attackers [151].

Therefore, in order to properly secure this layer and defend against attacks, the following are to be implemented: suitable authentication mechanisms, end-to-end authentication, key agreement schemes, PKI, Wireless Public Key Infrastructure (WPKI), secure routing and intrusion detection. Also, due to the increased amount of information circulation in the network, availability is also an important property that must be considered [151].

3.3.3 Application layer security

This layer sits at the top of the IoT architecture and corresponds to the services and application, together with the interfaces that allow the user to interact with the IoT systems. Due to the great variance in the number of applications, the secure environment may also have different requirements. Common problems affecting this layer are data privacy, access control and disclosure of information [130].

In order to secure the application layer, key agreement, authentication, database encryption and recovery solutions and the protection of private information using cryptography schemes are necessary [130].

In [151] two types of security measures are considered: technical and non technical. The technical side includes network authentication and key agreement across heterogeneous network,

Layer	Attacks/Issues
Perception	Node capture Fake node and malicious data Denial of Service attack Timing attack Routing threats Replay attack Side Channel Attack (SCA) Mass node authentication
Network	Illegal access networks Information eavesdropping Confidentiality damage Integrity damage DoS attack MITM attack Virus invasion Exploit attacks Network congestion Node authentication Privacy disclosure
Application	Data access permissions Identity authentication Data protection and recovery Handling of mass-data Software vulnerabilities

Table 3.1: Security issues and attacks affecting IoT architectural layers [151])

which could be based in symmetric key, public key (including PKI) and certification transfer technology. It also encompasses the protection of private information by use of fingerprint technology, digital watermarking, anonymous authentication and threshold cryptography. On the non technical side, increasing the awareness and sensibilization of users of general safety concerns, teaching them the importance of the overall security and how to properly use the IoT service. This could greatly reduce the leakage of confidential information for example. Also, on the non technical side, resource management, physical security information management and correct password management is considered as security measures to be in place in the IoT system.

A categorized overview the of IoT attacks and issues based on which architectural layer they may occur is is thoroughly presented in Table 3.1[151].

3.4 IOT FRAMEWORK ARCHITECTURES AND SECURITY

Since developing applications for the IoT is a challenging task, in recent years there has been a rise in the available frameworks specifically targeted towards the IoT domain. A framework may be defined as a series of guidelines, standards and protocols that are meant to aid

developers in the implementation of their applications. Without these guidelines, developers usually face the following challenges, as presented in [10]:

1. High complexity of distributed computing
2. Lack of frameworks or general guidelines to handle low level communication and to simplify high level implementation
3. Diverse range of available programming languages
4. Multiple available communication protocols

This way, the success of the IoT application is only as strong as its underlying framework and, therefore, the analysis of its security and privacy schemes are fundamental. In this section is presented a brief overview and discussion on some of these frameworks/standards and their security schemes.

3.4.1 Azure IoT Suite

The Azure IoT Suite [86] is Microsoft's framework for IoT development. It is an aggregation of services that allow users to interact with their IoT devices, visualize the generated data and perform big data operations, which are available through the Business Intelligence (BI) service [87].

Regarding the Azure IoT architecture, it involves the following entities:

1. **Azure IoT Hub:** consists of a web server which is in charge of the communication between the devices and the wide array of services available in the cloud. It handles all the security requirements of the system and has a registry containing authentication and identification information of the devices.
2. **Devices:** the devices are divided in two classes: Internet Protocol (IP)-capable and Personal Area Network (PAN) devices. IP-capable devices communicate with the Azure IoT Hub directly, through a supported communication protocol (Advanced Message Queuing Protocol (AMQP), MQTT or HTTP). However other protocols may be used, in which case the devices interact directly with a mediator entity, the Azure IoT protocol gateway, thus increasing the interoperability and underlying heterogeneity of the IoT system. As for the PAN devices, these are constrained devices which are not capable of running secure HTTP sessions and must rely on an additional gateway, the Field Gateway.
3. **Azure IoT protocol gateway:** gateway responsible for supervising devices, both IP-capable and PAN, which communicate using a different protocol than those supported by default (AMQP, MQTT or HTTP).
4. **Field Gateways:** entities responsible for receiving and aggregating all the PAN devices originating data, followed by its storage and forward, in a secure way, to the Azure IoT Hub.
5. **IoT solution backend:** corresponds to the large set of Azure cloud services.
6. **Presentation layer:** provides tools for data visualization.

When it comes to the security of Azure IoT, it is performed in a multi-level approach. At the device level there is device provisioning and their authentication where each device is provided with a unique key used to secure the communication between the device and the

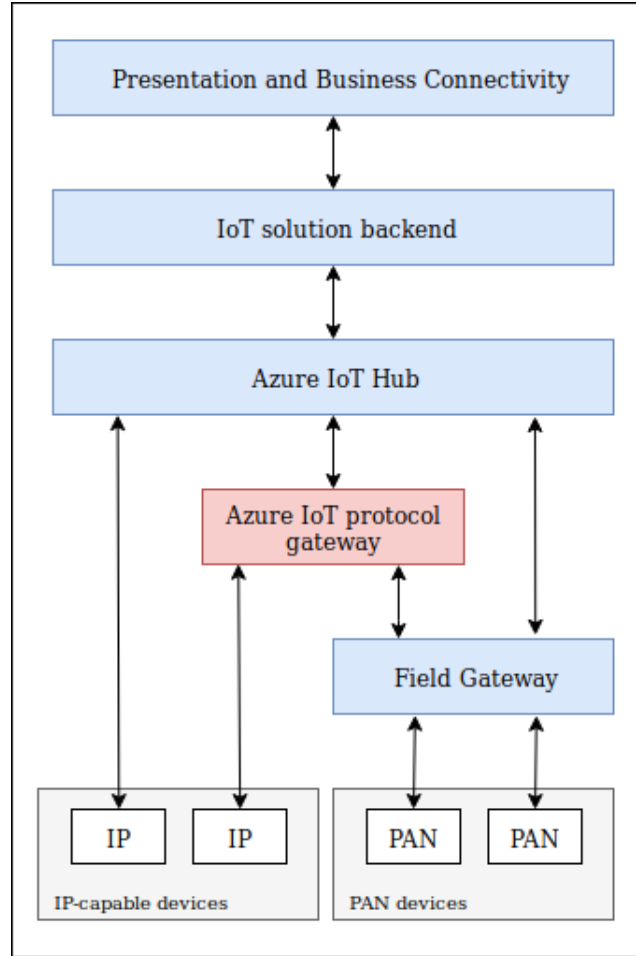


Figure 3.2: Azure IoT Architecture (adapted from [10])

Azure IoT Hub. Mutual authentication is required between the devices and the Azure IoT Hub. As such, the Azure IoT hub issues a unique device identity key for each of the devices during the deployment stage and the devices issue a token containing an HMAC-SHA256 signature to the Azure IoT Hub [10]. At the connection level it employs X.509/TLS-based handshake and encryption to secure the communication between the Azure IoT Hub and the devices and gateways. Also at this level, in order to protect devices from undesired incoming connections, it's the device responsibility to initiate and open all the connections with the Azure IoT Hub and not the opposite. Lastly, at the cloud level, the security takes advantage of solution accelerators which help keep data secure. Additionally, it makes use of the Azure Active Directory (AAD) which provides a policy-based authorization model for the cloud stored data, allowing for easy access management for posterior audits and reviews [118].

3.4.2 AWS IoT

AWS IoT [8] is Amazon's cloud platform for IoT development. It allows clients to connect their devices to the AWS or any other devices, while processing and actuating upon the data, and also guaranteeing its security. One distinguishable feature of AWS IoT is that

it enables applications to communicate with devices even when they are offline by using a specialized entity named “Device Shadow”. Its architecture consists of four major entities: Device Gateway, Rules Engine, Registry and Device Shadows. The Device Gateway is a mediator between the devices and the services in the cloud which communicate through MQTT. The Rules Engine is responsible for the processing and transformation of published messages and their delivery to their destination which corresponds to other devices that subscribed to the service. As for the Registry, its duty is to assign a unique identifier to every connected device, while the Device Shadow is a virtual image of an instance of every device. This image, also named shadow, is stored in the cloud and represents the last state of a device before being offline.

Regarding security, it is performed in the AWS IoT in a multi-layer fashion. It includes encryption and access control mechanisms and also offers a service to continuously monitor and audit security configurations [8]. AWS IoT ensures that authentication is performed not only when a new device is connected but also in every connection event, keeping track of the identity of the data originator. As such it verifies identities through one of three mechanisms, allowing more flexibility than other platforms: (i) X.509 Certificates, (ii) AWS Identity and Access Management (IAM) users, groups and roles [9] or (iii) AWS Cognito identities [6]. It is up to the end-user to choose the authentication mechanism. As for the authorization and authentication validation, this is performed by the message broker of the system. The authorization process in AWS IoT is based on policies which can be associated to each certificate or simply done by the use of IAM policies [7].

Regarding the confidentiality of the communication, it is achieved by use of SSL/TLS protocols, which are used to encrypt the connection between the device and the message broker. Additionally, AWS IoT supports Forward Secrecy (FS) which guarantees that the eventual compromise of long term keys does not affect the knowledge of temporary session keys. This is beneficial since if an attacker is able to extract the private key of a certain device, he is not able to decrypt any of the underlying communication messages, unless, of course, he learns the key used for such session [80].

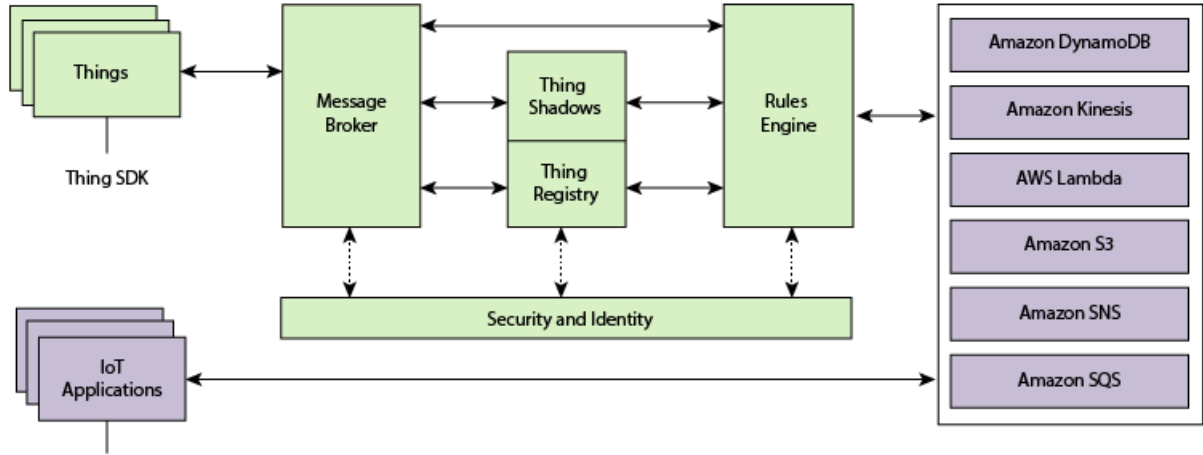


Figure 3.3: Amazon Web Services IoT Architecture [105]

3.5 BLOCKCHAIN SECURITY SOLUTIONS FOR IoT

In the recent years there has been a lot of discussion and research about the incorporation of blockchain technology as a mechanism to protect and solve many of the IoT security issues as an alternative to traditional PKI based solutions.

Simply put, a blockchain is a decentralized, distributed and immutable database, which is shared among the participants of a peer-to-peer (P2P) network and stores records of transactions and assets. As such, and as presented by the authors in [65], the application of blockchain to IoT could potentially solve issues such as address space, identity of things and governance, data authentication and integrity, authentication, authorization and privacy and secure communications. However, these solutions will not be further discussed here as it would shift from the scope of dissertation which does not consider blockchain for its final solution.

Authentication in IoT

As seen in the previous chapter, the authentication process plays a critical role in securing the IoT, being a requirement that extends through all of its architectural layers. This way, it is of the utmost importance to understand what exactly is authentication and how it can be applied to the IoT, which, as already seen, makes it a hard environment for the adoption of traditional authentication and security schemes. This constrained environment makes it so that proposed protocols must be lightweight and consider a trade-off between security and resource consumption. This also includes the communication overhead, since the number of exchanged messages on the authentication procedure must be kept to a minimum, as must the size of the messages be as compact as possible. Additionally, the protocols must take special care in protecting the location and identity privacy of both devices and end-users.

The base idea of authentication is to secure the transmitted data in order to restrict its access or tampering by an unauthorized party, while guaranteeing that the source entity in the communication is who it claims to be by validating its identity [13].

The first existing mechanisms of authentication were deeply dependent on secrecy, as demonstrating the knowledge of a certain secret was used as a proof of identity. However, with developments in the cryptography field, along with the introduction of digest functions and digital signatures, secrecy in authentication is no longer necessary in the current digital environments. As such, in order to better understand how authentication can be performed, it is important first to grasp some of the fundamentals of cryptography, as described in the following section.

4.1 CRYPTOGRAPHY FUNDAMENTALS FOR AUTHENTICATION

Regarding cryptography, it is performed through the use of specialized algorithms known as ciphers, that enable both encryption, to obtain a ciphertext from a given key and plaintext input, and its reverse operation, decryption.

The key is a group of bits used as an input parameter of a cryptography algorithm and its length is directly associated with the strength of such operation. Additionally, the requirement

of key secrecy depends on the mode of operation of the cryptography cipher, which can be either symmetric key or asymmetric key.

4.1.1 Symmetric key cryptography

In symmetric key cryptography (see Figure 4.1), the ciphers use the same key for both the encryption and decryption operations and work in two distinct modes: stream ciphers and block ciphers.

Stream ciphers take a plaintext input and combine (using the Exclusive Or (XOR) operation) each of its bits, one by one, with the bits of a keystream, computed by a pseudorandom cipher generator, resulting in the ciphertext. Both the encryption and decryption operations are performed under the same key (keystream).

Similarly to the stream ciphers, block ciphers also use the same key for performing encryption and decryption operations. However, the difference here is that the operations are performed on a larger group of bits (e.g. 128 bits) with a fixed length called the block, and usually apply one of two encryption/decryption modes [143]: Electronic Codebook (ECB) or Cipher Block Chaining (CBC).

One of the best applications of symmetric key ciphers is when encrypting large amounts of data, as they prove to be computationally efficient. However, one of the hurdles considering this scheme is that, when having to send encrypted data to another entity, both the sender and recipient must know the key value, implying the adoptance of a secure scheme for key distribution.

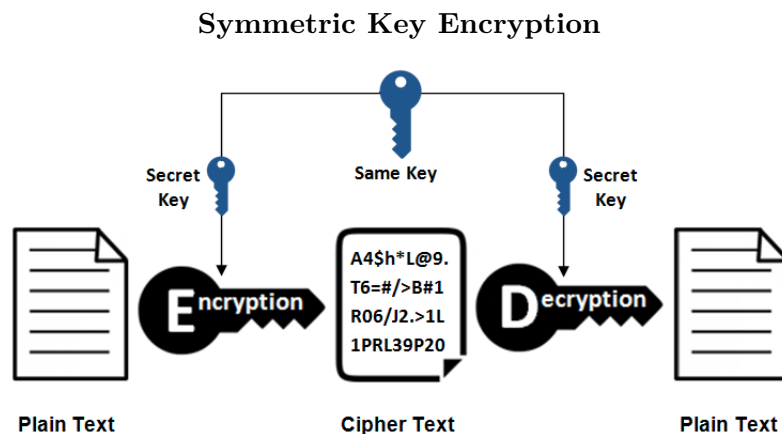


Figure 4.1: Symmetric key encryption [124]

4.1.2 Asymmetric key cryptography

Asymmetric key cryptography, also known as public key cryptography, makes use of a pair of keys for its encryption and decryption operations: public key and private key. The public key, as the name suggests, is a key that is widely available and distributed, and its knowledge represents no security danger whatsoever. In most schemes, the public key is distributed within a digital certificate, which is a signed document that guarantees that the private key

corresponding to the public key within the certificate is owned by its associated subject. Such bond is crucial for performing authentication and also assuring non-repudiation, since it allows the identification of the owner of the private key.

These certificates are issued by a Certification Authority (CA), a trusted third party that allows for the recipient of a certain certificate to validate its authenticity. The private key, however, is known only to its respective owner and cannot be publicly disclosed. As such, in the encryption operation anyone can encrypt any data using a certain public key. However, in order to decrypt and obtain the original content, this can only be done using its corresponding private key, meaning only the key pair owner is capable of doing so (see Figure 4.2).

Additionally, the asymmetric key ciphers also allow to perform what is known as a digital signature. Since the private key is supposed to be unique and not disclosed to others, by performing an encryption operation with such key, the resulting ciphertext acts as proof of identity that can be validated by anyone, through the use of the corresponding public key. Therefore, when sending a message to another entity, it is common practice to hash such message (for better performance) followed by the encryption of the resulting hash using the sender's private key. The produced signature is sent alongside the original message, allowing the recipient to validate its origin.

One of the major drawbacks of asymmetric cryptography is that its operations are computationally expensive, which is not optimal when dealing with large amounts of data (reason why the hash of the message is used to create the signature in the previous example).

Also, it is possible to adopt a hybrid scheme using both symmetric and asymmetric encryption when looking to send large data volumes to another entity owning a key pair. Such method is performed by encrypting the data using a symmetric key, generated on the fly. This key, with smaller size than the data to be sent, is then encrypted using the recipient's public key. This way, it is possible to take advantage of the benefits provided by both cryptography schemes.

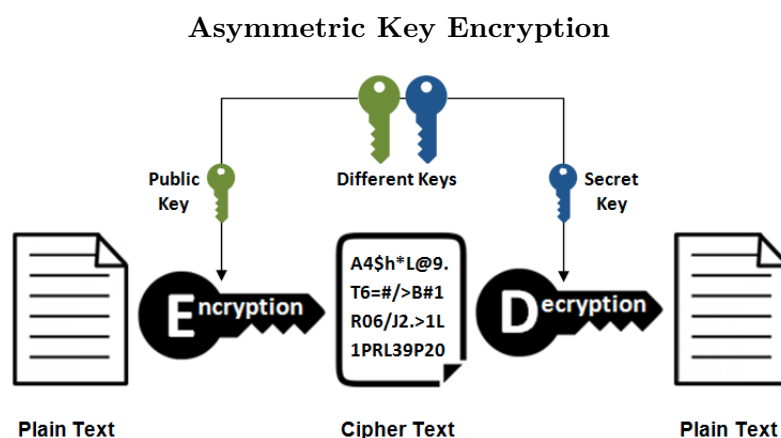


Figure 4.2: Asymmetric key encryption [124]

4.1.3 Cryptographic Hash Functions

Cryptographic hash functions, or digest functions, are one-way functions that do not require any keys, thus are not considered cipher algorithms, and that take a variable-length input to produce a fixed-length output, the digested value, acting as a unique compressed representation of its input, also known as data fingerprint (see Figure 4.3). A cryptographic hash function must guarantee three properties:

- Must be resistant to the discovery of the original text that produced a hash or digest value, meaning that it must be computationally expensive and complex to do so.
- Must be resistant to the discovery of a second text that produced a hash or digest value, meaning that it must be computationally expensive and complex to discover a new input value which its digest equals the digest of another given input.
- Must be collision resistant, meaning that it must be computationally expensive to find a collision, that is, to discover two inputs that produce the same digest.

These properties make digest functions suitable for authentication procedures, namely digital signature generation and validation, calculation of message authenticators and key derivation based on text keys or master keys.

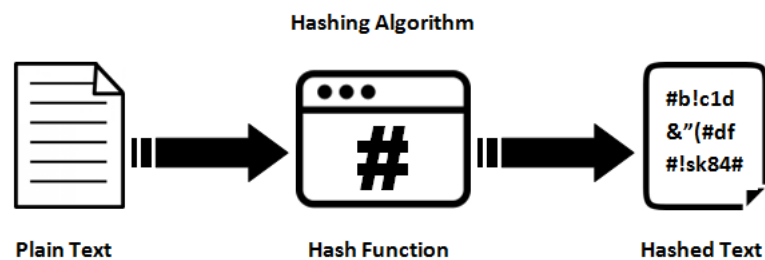


Figure 4.3: Hashing algorithm [95]

4.1.4 Message Authentication Code (MAC)

A Message Authentication Code (MAC) is a short value obtained from the application of a MAC algorithm that transforms a given message using a secret symmetric key, shared between the message sender and its recipient. As such, the MAC can be used to authenticate the message it was originated from, since only the entities that possess the secret key were able to generate such MAC, thus guaranteeing its authenticity. Furthermore, it also guarantees the message integrity, since any tampering to the original message would result in an incorrect MAC. Upon receiving the message, the recipient calculates a new MAC using its key and checks if it is equal to the one calculated by the sender, in which case the message is classified as authentic. Note, however, that a MAC does not allow for origin authentication towards any third party entity, since both parties could have generated a given message, as they both know the key for producing that MAC.

4.1.5 Digital certification

As seen in Section 4.1.2, public keys can be distributed within digital certificates that are issued by trusted third-party entities known as CAs, which, in turn, provide credibility to the certificates by signing them. This way, when an entity sends its public key certificate to another, the receiver, in order to validate it, must first validate the signature it contains, implying that it must obtain the public key of its CA, which in turn is also distributed within the public key certificate of the CA itself. Additionally, these CAs certificates are also issued by other CAs, which results in successive validation procedures forming a hierarchy known as a certificate chain. The CA at the top of every chain is known as the Root CA, and its public key certificate is signed by itself. Therefore, the validation of a public key certificate implies the validation of the entirety of its certificate chain.

Since the management of public keys and their certificates is not trivial, Public Key Infrastructure (PKI) were introduced as a practical solution for this problem. Simply put, a PKI can be seen as a set of CAs existent in a given context, along with their interconnection in certificate chains, and a collection of software/hardware mechanisms and policies for managing such certificates. This way, by adopting a PKI, interoperability is assured between applications requiring encryption and digital signature services.

4.2 AUTHENTICATION PRINCIPLES AND CLASSIFICATION

The classification of authentication schemes is done based on the security objective that one aims to achieve, which could be one (or more) of the following properties: entity authentication, message authentication, key authentication, nonrepudiation and access control [53]:

- Entity authentication assures not only the presence but also the identity of the claimant. The verification of its identity may be unidirectional, when it is performed by only one of the entities, or mutual (or bidirectional), when performed by both the sender and the recipient.
- Message authentication guarantees the origin and integrity of the information it carries to its corresponding receiver.
- Key authentication assures the bonding between an entity and its key. This process involves other relevant security notions, such as key management, key establishment, key distribution, key usage control and key life cycle. It is also at the key authentication process that CAs are responsible for asserting the authenticity of the keys, the bindings between keys and owners and the maintenance and revocation of certificates [64].
- Nonrepudiation ensures that an entity cannot deny the realization of a past action. A third party is often needed to resolve the dispute of deniability. This property is particularly important as its presence represents an undeniable proof that can be used as judicial evidence.
- Access control, also referred to as authorization, ensures restrictions upon an authenticated entity over certain data or resources of the system.

Authentication may also be classified based on the cryptography scheme used. In symmetric key based authentication, a shared key must be established between two entities who wish

to communicate with confidentiality, while the knowledge of the shared secret ensures the validation of such authentication. On the other hand, asymmetric key based authentication involves a third-party (or CA) that is trusted by the communicating entities and is responsible for assigning the entities' identity to public keys. The associated certificate can then be validated for authentication purposes.

However it is also possible to use asymmetric authentication without any third-party entity, as it occurs with Secure Shell (SSH), TLS and Internet Key Exchange (IKE), as well as use symmetric authentication adopting third-party entities, such as the Kerberos protocol.

As such, when considering the authentication of entities, we may consider the following parties in the process [53]:

- Claimant (or prover): entity that presents its identity, along with a proof of such identity, in a message, usually as a response to a previous one, in order to prove its genuinity.
- Verifier: entity that validates the received identity of the claimant
- Trusted third party: entity that acts as a mediator in the authentication process, offering an identity verification service that is trusted by both parties.

4.2.1 Authentication classification in the IoT

Unlike the traditional authentication schemes and protocols, the ones in the IoT domain are heavily restricted and must be developed with flexibility and scalability concerns. Additionally, due to the high heterogeneity of the IoT, they are also harder to classify, since additional aspects must be taken into consideration. In [47] the authors propose a taxonomy for the classification of IoT authentication schemes (see Figure 4.4) based on the following properties:

1. **IoT architectural layer:** As previously seen, a general IoT architecture is composed by three major layers: perception layer, a network layer and an application layer. Since they all require authentication procedures, these layers also have implications in the classification of the authentication schemes for the IoT.
2. **Authentication factor:** This classification criteria may be divided between identity and context. In identity-based authentication, the identity represents a piece of information presented by one part to another in order for it to authenticate itself towards the latter. Here the authentication is performed through the use of cryptographic algorithms, either symmetric or asymmetric, and hash functions. However, in context-based authentication, this process may be performed through the use of either physical or behavioral characteristics of individuals. The physical characteristics allow the extraction of biometric information to authenticate the users (also known as biometric authentication [22]), such as fingerprints, retinal scans or hand geometry. The behavioral characteristics, as the name suggests, are based on behavioral biometrics focusing on patterns that allow for the unique identification of individuals, such as keystroke rhythm, gait analysis (identification by the way individuals walk or run) or speech recognition.
3. **Architecture type:** Regarding the type of architecture used, the authentication method may be classified based on the granularity of the architecture. In a centralized architecture, an authentication procedure uses a centralized server, or trusted third

party, for authentication credentials' management and distribution. As for a distributed architecture, the authentication is performed in a straightforward manner between the communication entities. Furthermore, whether it is centralized or distributed, the authentication scheme may also be classified based on the hierarchy of the architecture, that is, if multiple architectural levels are involved in the authentication procedure, or the opposite, classified based on the absence of any hierarchy (flat architecture).

4. **Hardware-based:** The hardware of the device performing the authentication is also used as a classification criterion since this process might make use of its physical characteristics. As such, the use of hardware may either be implicit or explicit. In implicit hardware-based, the authentication makes use of physical attributes of the device hardware to strengthen the process, such as Physical Unclonable Functions (PUFs) [127] or True Random Number Generators (TRNGs) [129]. As for explicit hardware-based authentication, it leverages the use of a Trusted Platform Module (TPM), a hardware chip that securely stores and processes authentication keys.
5. **Token-based:** Tokens are digital pieces of information that aim to ease the authentication process by containing cryptographic data, such as keys, passwords or biometric data. This way, the authentication may also be classified by whether tokens are used or not. In a token-based authentication, identification tokens issued by a server are used to authenticate a user or device as it is the case with OpenID [100] and OAuth [49]. The Kerberos protocol also has a similar approach, as will be seen in Section 4.3.1.
6. **Procedure type:** Regarding the type of authentication procedure, authentication may be performed in three different approaches:
 - One-way authentication: only one of the parties in the communication authenticates itself towards the other.
 - Two-way authentication: both parties in the communication authenticate each other.
 - Three-way authentication: a central authority entity authenticates the communicating parties and acts as a mediator in their mutual authentication.

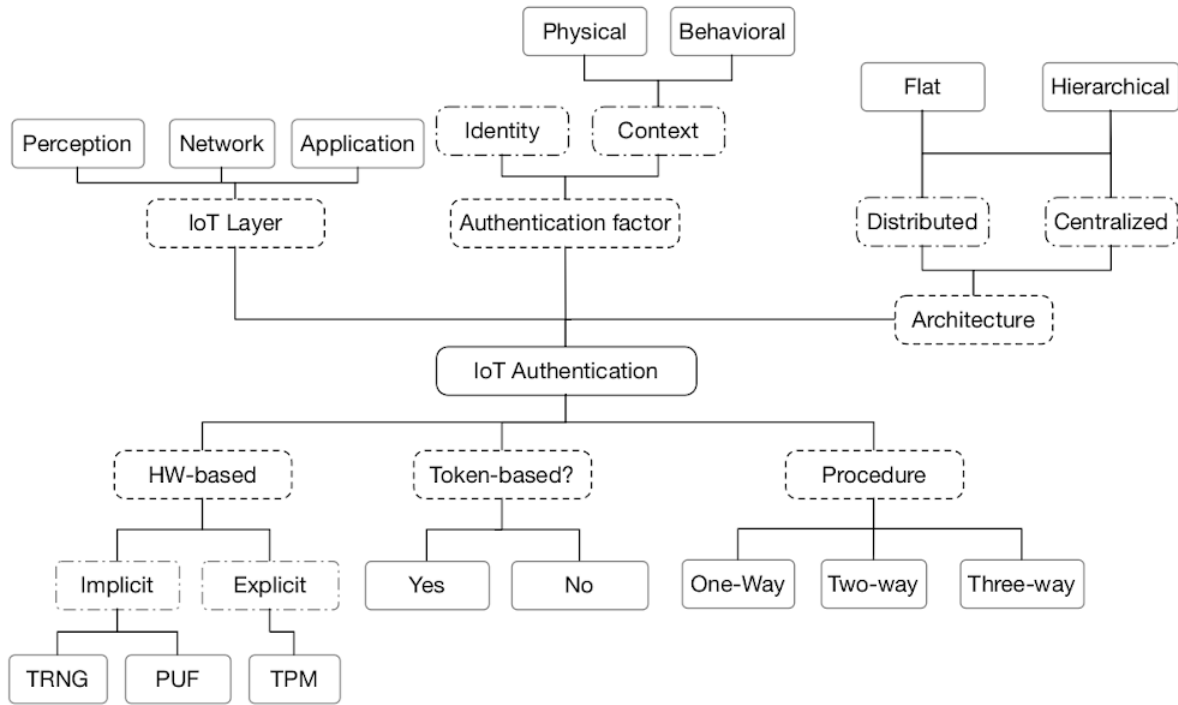


Figure 4.4: Taxonomy of IoT authentication schemes [47]

4.3 AUTHENTICATION PROTOCOLS

The following authentication protocols, although developed for standard computer systems, must be understood as their principles are in the basis of the developed solution in chapter 5.

4.3.1 Kerberos

Kerberos [68] is a network authentication protocol that was designed in the MIT during the 80s, conceived for the distributed system Athena, which aimed to connect the entire university campus with a collection of widely available services. During this time period, contrary to what is the norm nowadays, there was yet to be any asymmetric key cryptography schemes, using asymmetric keys and public key certificates, widely deployed and aimed at the authentication of people and services. Therefore, the Kerberos authentication process was built on top of shared, symmetric keys.

It follows the principles introduced by the Needham-Schroeder protocol [94], where a Key Distribution Center (KDC) is used to authenticate the interacting parties of a distributed system, usually clients and services, and to distribute session keys between them (and the KDC). Kerberos is, however, more complex than its predecessor, mostly due to the double functionality of its KDC, which includes an additional service called Ticket Granting Service (TGS). The TGS is responsible of emitting credentials, named tickets, that will be used to access the available services.

Kerberos operates based on two types of credentials, these being the ticket and the authenticator, which are used as identity comprovatives over an untrustworthy network

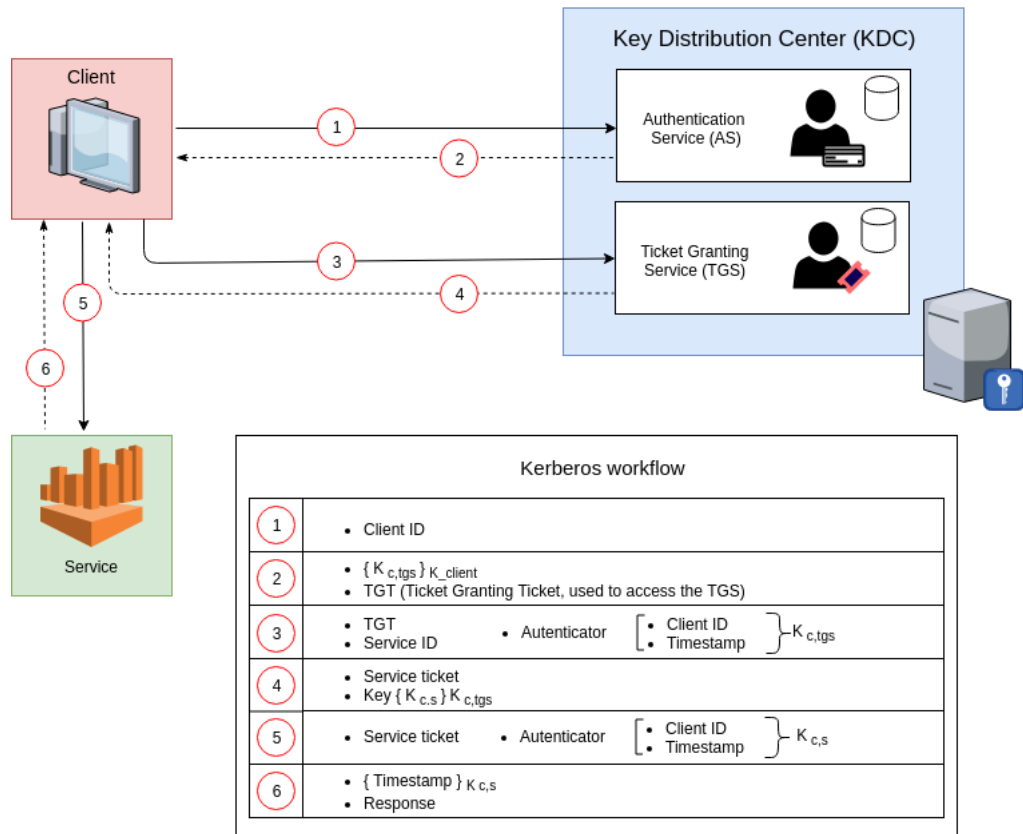


Figure 4.5: Kerberos workflow (adapted from [17])

channel. Tickets are encrypted and emitted by the TGS and are used in two scenarios: (i) used by the client in order to access a specific service and (ii) used to distribute session keys to the servers so that they can authenticate the client. The general structure of a Kerberos (Version 5) ticket and authenticator, as specified in its Request for Comments (RFC) [68], can be seen in Figure 4.6. The ticket carries the session key generated by the KDC for the mutual authentication of client and service, client/server realms and names, a validity period represented by a start and end time and a collection of addresses, usually IP addresses, used for client applications to access the service in question.

The Kerberos authenticator is used, along with the ticket, so that a client can prove his identity when accessing a service and also for the target service to verify if the request is not a repetition of a previous request from the same client. The authenticator carries the emitting server name and realm, and a timestamp with the time of its creation. Optionally it may contain a sequence number, used to detect the repetition of messages, and a subsession key used if a high amount of data is to be exchanged between the client and server, which would result in an overload of their shared session key. Figure 4.5 depicts the communication steps of "Kerberized" client access to a "Kerberized" service.

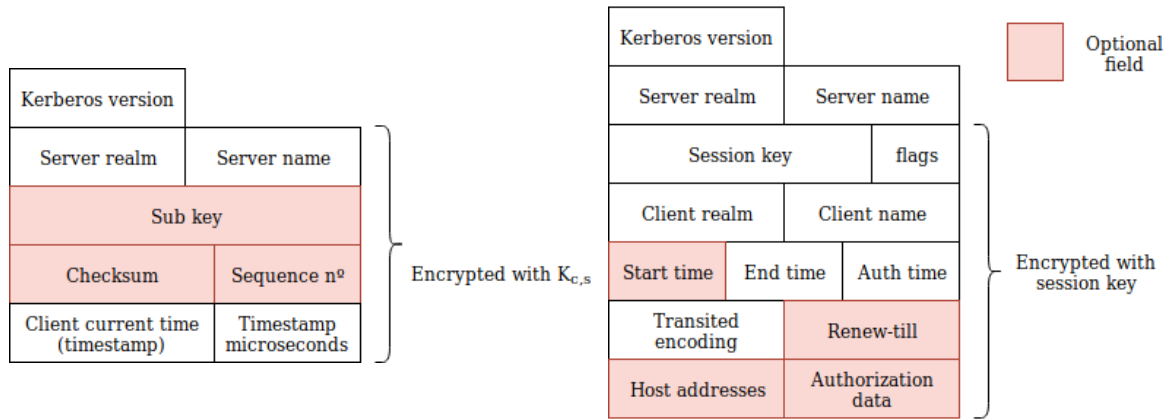


Figure 4.6: Kerberos authenticator structure (left) and ticket (right)

4.3.2 OAuth 2.0

OAuth 2.0 [49] is an open standard authorization protocol which allows online services, known as resource owners, to provide limited access to the resources they own. Such access is requested by third party web applications, known as the clients, and is performed without requiring any credential sharing between client and owner. Instead, the client is provided, by an authorization server, with an access token (with previous approval by the respective resource owner), which is then used to access the protected resource.

The OAuth 2.0 specification clearly defines four roles:

- Resource Owner: entity capable of granting access to a protected resource. Usually referred as end-user when the resource owner is a person.
- Resource Server: server hosting the protected resources. Capable of accepting and replying to the requests through access tokens.
- Client: the application requesting access to protected resources on behalf of the resource owner.
- Authorization Server: the server responsible for authenticating resource owners and issuing access tokens to the clients after obtaining authorization.

When a client requires access to a certain resource, it must first request authorization to the resource owner. If the request is authorized, the client will receive an authorization grant, a credential which represents the resource owner authorization. Following, the client requests an access token from the authorization server, by first authenticating and presenting the authorization grant. The authorization server will then authenticate the client and validate the authorization grant. If valid, the access token is then given to the client. Now owning an access token, the client requests access to the protected resource by the resource server, and authenticates by providing the access token, which is then validated. If valid, the resource server will serve the respective request. This interaction flow can be seen in Figure 4.7.

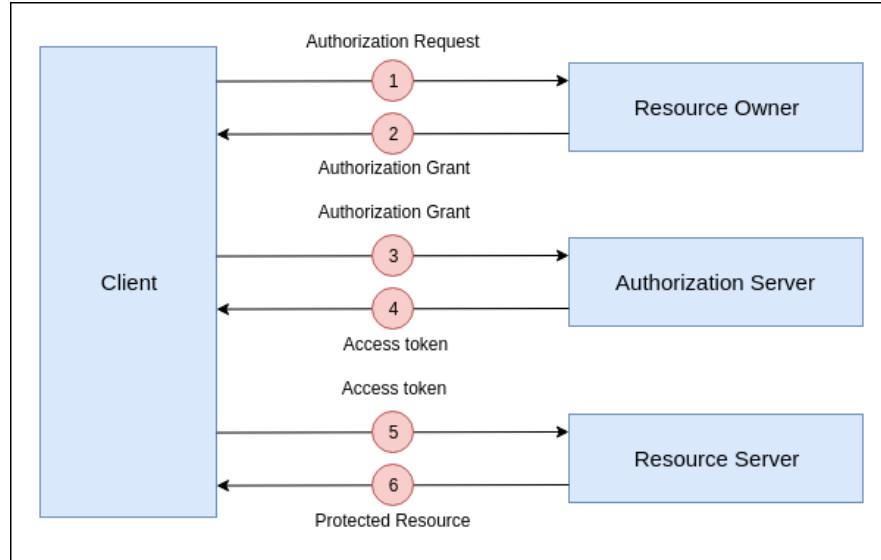


Figure 4.7: OAuth 2.0 Protocol Flow (adapted from [49])

4.3.3 eXtensible Access Control Markup Language (XACML)

The eXtensible Access Control Markup Language (XACML) [97] is an open standard that defines both an access control policy language and an architecture to apply authorization mechanisms. It was designed with the intent to provide interoperability and homogenize the access control schemes of different systems. From an architectural standpoint, the XACML model defines five communicating entities, allowing a clear separation of duties. The Policy Decision Point (PDP) is responsible for interpreting the access control policies and respond to the Policy Enforcement Point (PEP) with a verdict about a given access request. The PEP is located between the service requester, entity that wishes to access a service, and the service provider. The Policy Administration Point (PAP) is responsible for managing the access to the access control policies, while the Policy Information Point (PIP) acts as a source of attribute values, containing information complementary to the policies. The access policies are stored typically in a database or file system at the Policy Retrieval Point (PRP).

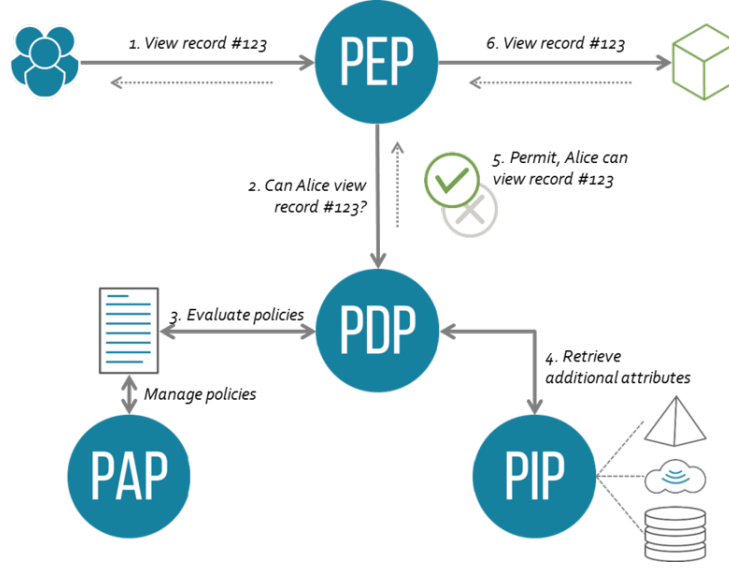


Figure 4.8: XACML architecture and message flow [16]

4.3.4 Lightweight cryptography solutions for IoT

Lightweight cryptography (LWC) is a recent state-of-the-art cryptography field that tackles the implementation issues of classical algorithms on constrained environments, such as RFID tags, sensors, smartcards and health-care devices, thus suitable for IoT scenarios [63]. As such, these algorithms aim to use less memory, less computing resources and less power supply than the classical adopted algorithms, at the cost of decreased security liability. Some examples of proposed lightweight algorithms are PRESENT, CLEFIA, LED, KANTAN, etc. [24]. In 2013 NIST started a lightweight cryptography project in order to study the performance of current approved algorithms on constrained devices, understand the need for such algorithms and plan their possible standardization [84]. Although lightweight algorithms seem promising for securing the IoT and authenticating its devices, they are still in their infancy and, as such, were considered out of the scope of this dissertation.

IoT Security Architecture

In this chapter, we present the architecture developed for securing IoT environments, along with its progress rationale and decision-making, which resulted from the numerous meetings in regards to the subjacent academic research project (Angerona) and throughout the development of this dissertation. As such, the entities of the architecture and the authentication protocols that came from it, that is, the network bootstrap configuration, the session setup and authentication ticket fetching are here thoroughly presented and described.

5.1 TOWARDS A FIRST ARCHITECTURE DRAFT

5.1.1 Desired security objectives

When first brainstorming about the architecture to be developed for securing a generic IoT environment, we had the primary objective of creating a system revolving around the access control to the devices and their data. That meant preventing undesired access and manipulation by attackers, as well as preventing these IoT devices from leaking information to non-authorized arbitrary endpoints.

Additionally, since it is expected that most IoT devices operate without any supervision, it is crucial to understand that any security credentials to be stored in the devices are virtually vulnerable and prone to being accessed and extracted, thus compromising the safety of the whole system. Therefore, the security credentials had to be defined in such a way that its knowledge by an attacker could not cause any disturbance to the IoT network. As a solution for this problem, we assign a public key pair to each component present in the IoT architecture, as opposed to any pre-shared and immutable symmetric key/secret. Furthermore, there is no need to use any PKI scheme, since the relevant public keys can be sent to the desired entities in a previous discovery step, similarly with the SSH protocol [82].

However, one last problem that needed to be addressed was that, as previously mentioned, due to the resource constraints the devices might have, the use of asymmetric cryptography must be minimized, applied only when necessary. This way, the remaining cryptographic operations must be performed through symmetric cryptography schemes.

5.1.2 The initial architecture draft

From these premises, it was clear from the start that the gateways bridging the IoT devices (essential entities in every IoT architecture, as seen in Section 2.6) would have a critical security role to play. Since these gateways are, in most cases, the only points of access between the IoT devices and the rest of the network, they are perfect candidates for mediating the access control to the IoT devices. As such, this would imply that a certain authentication mechanism, undefined at this stage, would be necessary in order to assure that the IoT devices only communicate with authorized gateways. In the same fashion, the users of the system, when looking to access a certain IoT device, would have to authenticate themselves towards the respective gateway(s) and vice-versa. This way, it would be possible for the clients to perform their access to the devices without ever directly interacting with them, thus increasing the privacy of the data they produce, as it could only be delivered to a client through the authenticated gateway.

However, having only the gateways securing the network was soon proven to be suboptimal and inefficient, especially when considering the scalability and management of large IoT networks, which is usually the case. This led to the creation of an additional entity to the architecture that would supervise and manage the gateways with similar access control policies, while mediating the authentication and authorization to the IoT devices. As such, the authentication was now intended to be performed through a redirection to this entity, which would be later denominated the IoT AAA server, responsible for performing such operations. As for the authentication protocol to be applied, it was decided that it would be in the form of authentication tickets in a similar fashion as Kerberos (see Section 4.3.1), being the AAA server the equivalent of a Kerberos TGS. However, it would have two differences: (i) our tickets would be based on asymmetric primitives, while the original Kerberos only uses symmetric ones; and (ii) Kerberos centralizes the ticket issuing in two servers (TGS and KDC) per realm (or domain) while we consider multiple per domain. Therefore, in order for a client to communicate with a certain IoT device, it would request a ticket from the respective AAA server and send this ticket along with its access request to the target gateway. The gateway, upon validating the ticket, would then grant access and, at last, relay the request to the IoT device and the resulting response back to the authorized client.

A representation of this architectural concept and its entities' interactions can be seen in Figure 5.1, which also includes a generic Internet gateway for remote environment communication with the IoT domain. In this depiction the IoT device, although not explicitly presented, is considered to be the aggregation of the sensors/actuators and the IoT driver, that is, the support environment (both hardware and software) that contains the appropriate technology to read and write data from the sensors/actuators and to communicate with its respective gateway(s).

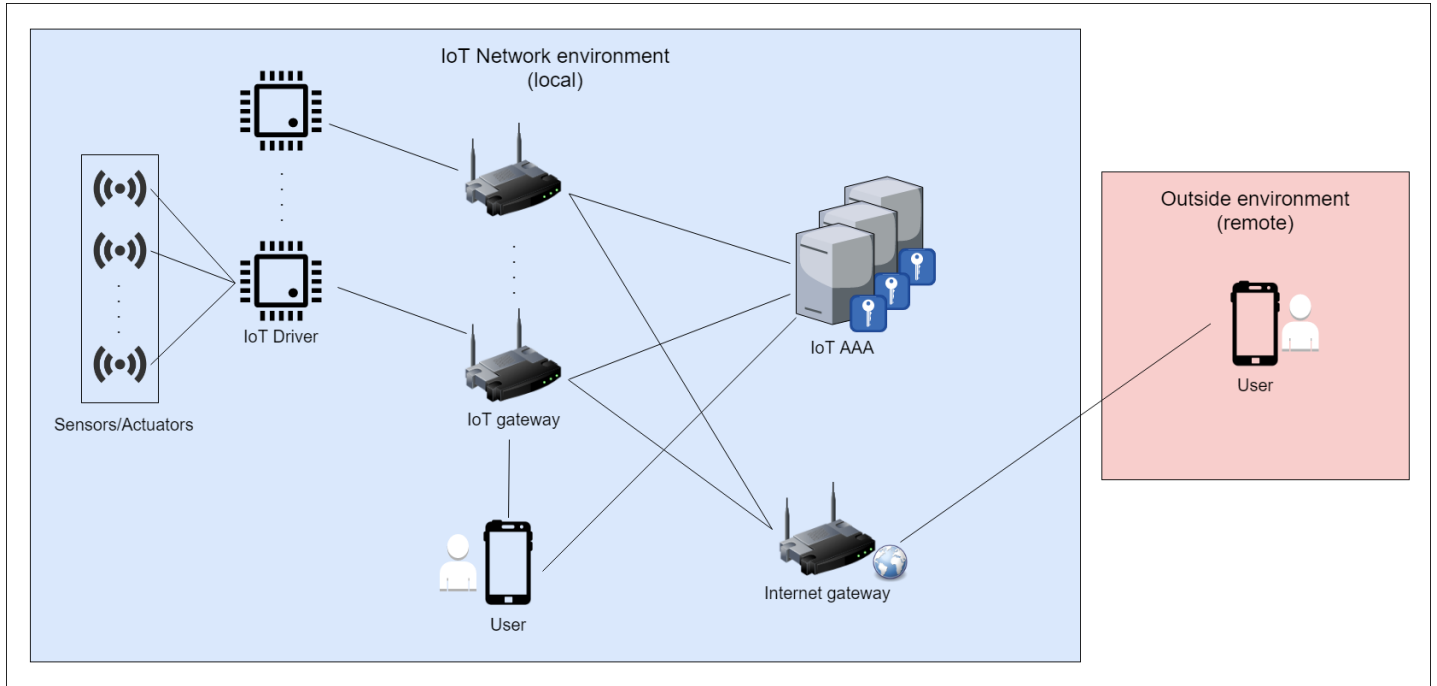


Figure 5.1: Initial draft of the IoT architecture

5.1.3 Towards a more reliable and robust security architecture

Although the initial architecture structure seemed feasible at first glance, it still faced some problems, namely with regards to scalability, flexibility and interoperability, i.e, when considering multiple applications running at the level of the IoT devices. Indeed, if the ultimate goal was to have a robust and secure IoT architecture, the system would have to incorporate different IoT sensors and actuators from several manufacturers in such a way that any existing conflicts between them could be avoided. Additionally, having a single IoT AAA server entity centralizing and managing all of the network entities would later be proven inefficient, as the inevitable growth of the IoT network would ultimately overload the server. Although the network may (and should) have multiple IoT AAA servers, especially for avoiding single points of failure, it would still centralize too many tasks, reducing drastically the flexibility the network would allow, unlike the new architecture. As such, it was a clear necessity that a separation of duties regarding the operations performed by IoT devices had to be considered along with their security and authorization provisioning.

This way it was decided that the old IoT driver and the sensors attached to it would now be divided into three different, but interconnected, components: the IoT sensors/actuators, the software code, or drivers, that control the sensors and, finally, the physical host device that bridges the latter and the former by running the drivers, through which the sensors are connected. As such, the terminology of the network entities would also have to change: the IoT driver now represented the support code for connecting to the IoT device/actuator, later named the Device Driver (DD) (which was previously not considered as a separate entity)

and the old IoT driver, that is, the host device, would now be entitled Device Host (DH). Having divided the duties at the level of the device, in this new decentralized scenario the security provisions, namely the authorization to access an IoT network device, would still be performed through AAA servers, using a concept of enforcement similar to that of XACML (see Section 4.3.3). Therefore, and in a similar fashion with the initial authentication protocol draft, and in order to complement the new entities, each DH and DD would be linked to a supervisor AAA server in the IoT network. Additionally, each network Gateway would also have a bonding to a supervisor AAA server, mediating the access between them and other entities in the network.

5.2 FINAL ARCHITECTURE SPECIFICATION

As mentioned before, this new architecture allowed the establishment of a clear separation of the entities providing the services from those which authorize the access to the former. As such, it is comprised of seven different entities: IoT devices, Device Hosts (DHs), Device drivers (DDs), IoT Gateways, AAA Controllers (A3C), DHM and Clients. These entities and their role in the IoT network will be thoroughly described in the following sections. However, at the time of writing this dissertation, the architecture only considered operations within a local environment, i.e, local network. Therefore, all the remote access to client applications and cloud services, the case with many IoT applications, was not here taken into account and was contemplated as future work. Figure 5.2 depicts this new architecture and the communication flows between its entities.

5.2.1 IoT devices

Unlike in the previous architecture, here we consider the IoT device to be only the sensor/actuator part, excluding the hardware supporting the remote communications. As such, they are the smart “things”, mentioned in the previous chapters, as they are able to sense or act in the physical environment. Therefore, they are only capable of performing these simple actions and thus have no notion of security and cannot communicate with any outside endpoint. The only communication they perform is through a local hardware bus/link (Inter-Integrated Circuit (I2C), Serial Peripheral Interface (SPI), UART, etc.) with a master device that supports it. This host device is here known as the Device Host (DH). Each DH runs what is known as a Device Driver (DD), which is a dedicated software driver for reading and interpreting the IoT device generated data.

Also, in addition to the previous architecture, here each IoT device is bound to an AAA controller server (A3C), the entity of the architecture responsible for defining the access control policies the IoT devices must enforce relative to its Clients. This enforcement, however, is not performed by the IoT device itself (since, as mentioned, they have no processing capabilities besides their sensing/acting purpose) but by their respective DDs running in the DH. This authorization (see Section 5.3.4) is performed in the form of a ticket produced by the IoT device A3C server and given to the Client aiming to access the device. The ticket is then

presented to the DD (through the respective GW) which, if valid, initiates a session with the Client, fulfilling its request.

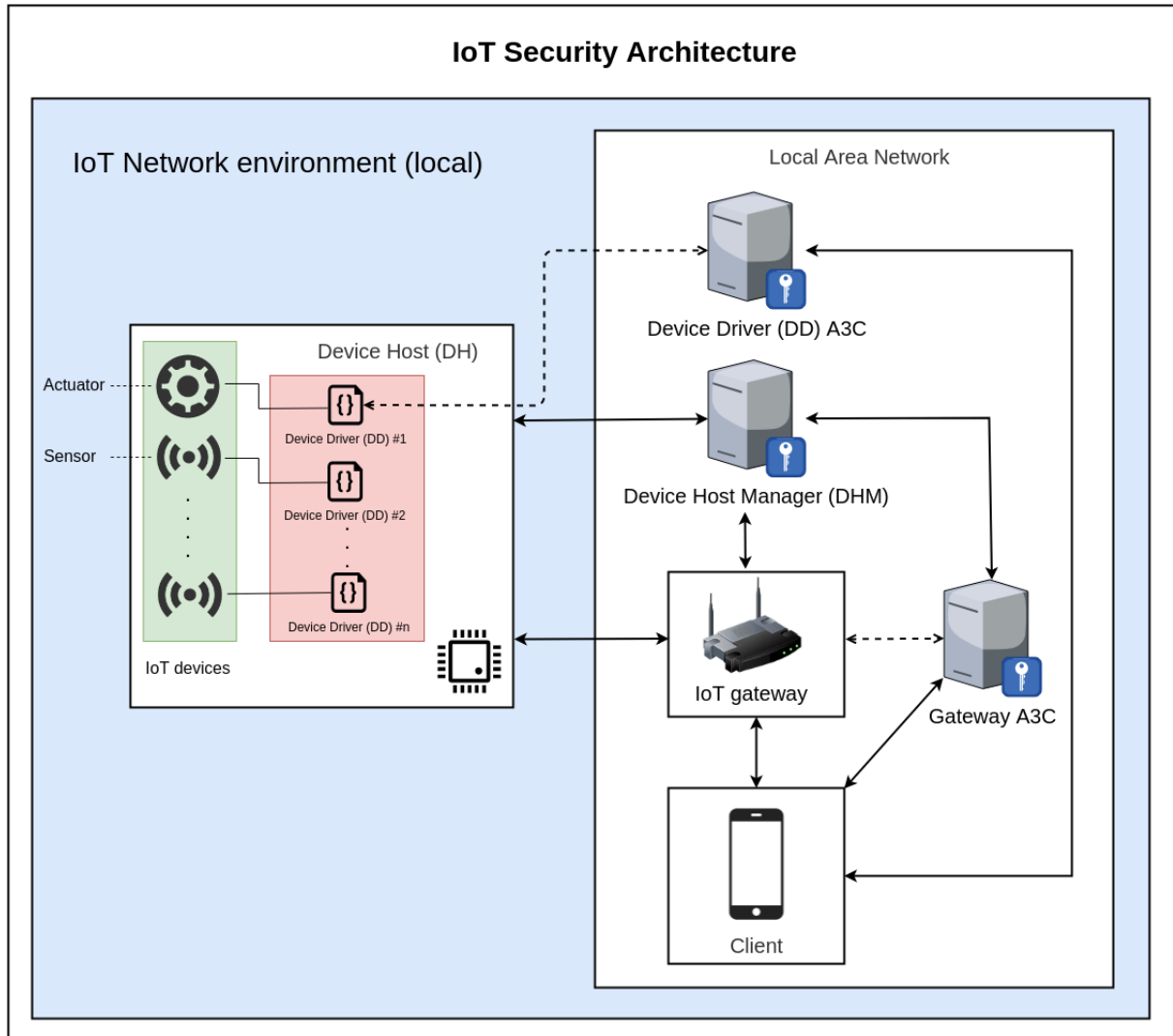


Figure 5.2: Developed IoT Security Architecture

5.2.2 Device Hosts (DHs)

The Device Host (DH) is the entity that is able to connect and control the operation of one or more IoT devices. It is a low-power computational system with communication capabilities, thus allowing the establishment of a connection with IoT Gateways, one at a time. This way, it creates a multi-hop and end-to-end communication with other network entities, i.e, the Clients of the system. This link would, in the majority of the appliances, employ any low-power wireless transmission technology, for instance Bluetooth or ZigBee. However, cabled transmission is not in any way excluded, since it is still required for applications with higher bandwidth throughput (video streaming services, for example). As previously mentioned, each DH runs multiple applications, the DDs, that control the connected IoT devices. Additionally,

the DH itself runs a primary managing application that is responsible for supervising the DDs, loading and executing their respective code, and is also responsible for all the remote communications with the Gateway. Each DH is supervised and managed remotely by an entity in the network known as the Device Host Manager (DHM). This management includes the mapping of which Gateways will be allowed to establish a connection with each DH, done by the provision of a configuration ticket during the setup stage of the IoT network (see Section 5.3.5).

5.2.3 Device Drivers (DDs)

A Device Driver (DD), unlike the other architectural entities, has no hardware involved, as it is just generic code that executes on top of the DH managing application, controlling one of its IoT devices.

Each DD defines a high-level interface, or API, with a specific set of functions that the DH uses to operate and interpret the data of the respective IoT device. These functions are enforced locally by the DD, but only upon a remote request from an authorized Client in the network. Such Client receives clearance to the API functions by a previous access control step defined by the entity that manages the DDs, the DDs A3C server. From the DD perspective, it only verifies if its own A3C has issued the access ticket received from the Client, followed by the fulfillment of the request if valid.

Upon completing a certain action, the DD will relay the response, that is, the result of a sensor reading or the log of a certain actuator operation, exclusively to high-level communication endpoints (message queues) defined at the Gateway. Note that, from a communication perspective, the DD relays the response to the DH, which is then responsible, as previously seen, for the transmission into the appropriate Gateway queue.

Additionally, since the IoT device has no notion of security, all cryptographic operations are offloaded and performed by the respective DDs. As such, the DD is responsible for handling the access control of Clients and the confidentiality of the data produced by the IoT device. Therefore, the DD may implement encryption over the data, thus allowing end-end protection in the communications between the Clients and DDs against malicious Gateways or network eavesdroppers. This confidentiality property is set by the DD A3C at the moment of the DD deployment in the DH and is usually bound to the sensitivity of the data itself. However, in order to allow more flexibility, this property may also be set, or changed on the fly, in the communication establishment between the Client and the DD (see Section 5.3.6), where the Client fetches an access ticket from the DDs A3C. The confidentiality property is then set within the ticket, alerting the running DD of such notice when interpreting it.

From the perspective of an IoT device manufacturer, this method proves to be beneficial as they may create their pieces of generic code for controlling their property as DDs, resulting in extra interoperability and thus increasing the heterogeneity of the IoT network.

5.2.4 IoT Gateways

The main purpose of Gateways is to provide an indirect access for the different Client applications to the IoT devices through their DH and DD, respectively. As such, the Gateway

is the interconnecting agent/broker between the external IP network, composed of the Clients, A3Cs and DHMs, and the internal IoT network, comprised of DHs and their IoT devices, which are usually not IP addressable. This means that the Gateway must be connected to a local area network, for example through Wi-Fi, being accessible to a variety of devices such as laptops, desktops, tablets or smartphones.

From a software perspective, the Gateways run a set of services enabling Clients to find, not only those Gateways but also the IoT devices they connect to (discovery service), as well as services to interact and relay request/responses to/from the IoT devices (messaging service). The discovery service is not here specified (along with the discovery process of the remaining network entities) as it is out of scope for this dissertation, although a generic broadcast discovery mechanism would be enough for this scenario. The messaging service is implemented via queues that hold messages exchanged in the Client-DD data flow, which are processed according to a queuing discipline, such as First In First Out (FIFO).

Similarly to the first architecture model (described in Section 5.1.2), here the Gateway is also the critical agent that provides a first layer of protection for the IoT devices, filtering any undesired network connections. As such, and similarly to the DDs, each gateway is supervised by an A3C server, which grants Clients an access authorization in the form of tickets.

During the setup of the network, that is, when devices are discovered and the network topology is formed, the Gateways receive from DHM servers the information of the DH devices they must connect to, in the form of a configuration ticket. Additionally, upon establishing a secure session with the DHs, the Gateway will receive a set of attributes from the DDs that are installed in that respective DH. Such attributes are then provided to the system Clients that wish to communicate with the connected DDs. This way, the Gateways can be also seen as temporary stockpiles of a set of attributes: the DDs attributes, available to authorized Clients and their A3Cs and DHs attributes, available to the DHs DHM.

5.2.5 IoT AAA Controller (A3C)

An AAA (Authentication, Authorization and Accounting) Controller (A3C) is the architectural entity responsible for defining the access control policies for another entity in the network, namely the DDs (in regards to their IoT Devices) and the Gateways. This access is enforced and defined concerning the Clients, when wishing to access a remote IoT Device, or the DHMs, when looking to configure the respective network Gateways with the DHs they shall connect to (see Sections 5.3.2 and 5.3.6).

Therefore, we can see the A3C as an online server that receives access or configuration requests, authenticates such requests and makes a decision whether the entities sending them are eligible and authorized to access the entities supervised by the A3C. If this validation is successful, an access/configuration ticket will be issued and will be presented by the requester to the recipient entity it wants to access. Additionally, in the case of target DDs, the security policy within the ticket must contain a clearance decision for each of its API functions.

5.2.6 Device Host Manager (DHM)

Initially considered as an A3C for DHs, the Device Host Manager (DHM) differentiates itself from a simple A3C server due to its extended functionalities and role in the network. Instead of only granting access control to DHs and managing them, they are also responsible for configuring the Gateways that connect to the DHs they supervise by sending the necessary configuration parameters. Only with this configuration step are the Gateways allowed to discover and connect to the respective DHs. As previously mentioned, this setup operation is performed via a configuration ticket which the Gateway uses to prove, towards the DH, that it was allowed by its DHM to interact with it, resulting in the creation of a valid transmission link. However it is important to remember that, before this operation takes place, the DHM must first be authorized by each A3C of the Gateways it targets in order to start the DHM ↔ Gateway link.

The DHM may also provision the connection of multiple Gateways to the same DH, thus granting a higher degree of redundancy and flexibility to the overall IoT network. Therefore, we can consider the DHM to be the central entity of the architecture. Each DHM will store information about the DH devices they manage: their Identifier (ID), address (ex: Bluetooth), public key, list of installed DDs and the IDs and addresses of the Gateways they shall connect to.

5.2.7 Client

The Client can be seen as the application that makes use of the IoT devices deployed in the network. However, in order to interact with the devices, the Client has to acquire two authorization tickets: one to access the Gateway that connects to the target device (provided by the Gateway's A3C) and another to access the device's DD (provided by its A3C). Any storage and posterior analysis of the data originated by the IoT devices is considered to be managed by the concrete Client application and is not considered in this architecture specification.

5.3 ARCHITECTURE AUTHENTICATION PROCESS

As previously mentioned, the access to devices is always performed through the emission of authorization tickets by their managing device, either DHM or A3C. The tickets, upon validation, are then used as proof of authorized access by the target device and used in order to create secure sessions on the fly. In this section, the generic structure of the tickets is specified, along with the session setup protocol and the network setup and configuration procedures.

5.3.1 Configuration/Authorization Ticket structure

Each ticket used in the architecture possesses the same three parted structure: a secret or private part, a public part and a signature part.

The secret part contains a confidential master key used for the establishment of secure sessions with regards to data encryption and integrity. When the issued ticket is received and

validated by its target entity, the session key will then be used for securing such session until it expires. The ticket private part is always encrypted with the public key of the ticket target (K_{target}^+).

The public part contains all the data in the communications procedures involving the ticket that do not require any sort of confidentiality. This usually includes the ticket target identifier (ID), a pseudonym of the ticket owner, the ticket expiration date and the set of rights the ticket owner has over the target.

The signature part contains the signature of the ticket and is performed by the entity issuing it over the other two parts, secret and private. The signature is computed with the private key of the ticket issuer (K_{issuer}^-).

5.3.2 Network bootstrap and configuration

When the bootstrap setup begins, we assume that all devices know the routes to each destination, i.e, a certain discovery protocol was executed in a previous step that allowed, for all the entities of the architecture, to know each other's presence and their location in the network. That said, it is also considered that clients already have full knowledge of the available IoT devices in the network and their underlying APIs. This way, the initial state of the network is as follows:

- Each A3C server has an asymmetric key pair ($K_{\text{A3C}}^-, K_{\text{A3C}}^+$).
- Each Gateway has an asymmetric key pair ($K_{\text{G}}^-, K_{\text{G}}^+$) and the public key of their A3C.
- Each DHM has an asymmetric key pair ($K_{\text{DHM}}^-, K_{\text{DHM}}^+$) and a set of configuration parameters for enabling Gateways to connect to the DHs they oversee. Additionally, the DHMs also contain the DDs code that should run in those DHs.
- All DHs have their asymmetric key pair ($K_{\text{DH}}^-, K_{\text{DH}}^+$) and the public key of their DHM. Initially the DH is connected to the installed IoT devices but there are no deployed DDs.
- All Clients have their asymmetric key pair ($K_{\text{C}}^-, K_{\text{C}}^+$).
- All IP addressable entities, that is, DHMs, A3Cs and Gateways, are connected to the same local area network.

From this initial state, the network will then proceed to perform three sequential configuration steps in order to become fully operational.

First, each DHM will search for the gateways to whom the DHs, managed by the former, are meant to establish a connection. However, in order to access the target Gateways, first they must request an access ticket from the Gateway A3C server, as specified in the following section (see Section 5.3.4). Next, the DHM, now owning a valid access ticket, sends it to the target Gateway in order to establish a secure session (see Section 5.3.5). Upon the creation of a session, the DHM will generate one access/configuration ticket for each DH the Gateway must connect to.

This way, in the second step, the Gateway proves towards the DH that it is a valid entity, properly authorized by its DHM, by presenting the ticket followed by the establishment of a secure session.

Lastly, in the third configuration step, each DHM finds out which Gateway is connected to one of its DHs, checks the DDs installed in each of its DHs and installs the missing DDs. In this step the DHMs can be alerted by the Gateways upon their successful connection to DHs.

This bootstrap process can be seen in Figure 5.3.

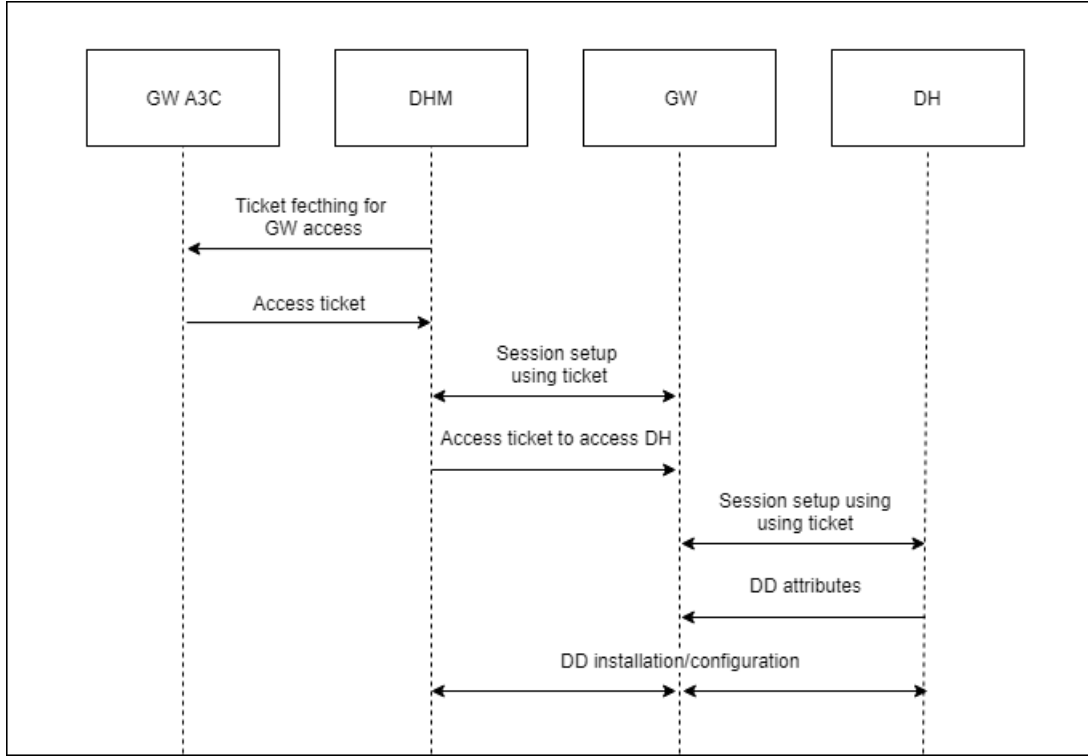


Figure 5.3: Network bootstrap and configuration protocol

5.3.3 Identification and naming

5.3.3.1 Universal Unique Identifiers (UUIDs)

As mentioned in the previous section, all the components of the architecture, with the exception of the IoT devices, have an associated asymmetric key pair, from which its public key is used as their unique identifier in the IoT network. However, since public keys can be rather large bit arrays, a shorter implementation of UUIDs was adopted, using the 128-bit MD5 digest of the public key value.

The choice for MD5 as the digest algorithm was due to its fast performance [15], advantageous in constrained DHs and Gateways. Although its use is banned from cryptographic operations requiring collision resistance, it can still be used for UUID generation since finding a collision for a given public key is not only extremely hard [136] but is also of no use for an attacker, as it still needs to have the corresponding key pair in order to prove the ownership of the UUID.

5.3.3.2 DDs/IoT devices attributes

The name of an IoT device is provided both by its DD and its corresponding A3C and the attributes they yield are each signed with their private key (K_{DD}^- and K_{A3C}^- , respectively).

As for the DDs, they define several operational attributes for the devices they manage. Some of these attributes may be provided by the IoT device itself as, for example, the manufacturer name and device model, considered of optional character. The remaining attributes correspond to the DD itself, containing both optional and mandatory values. Optional values can be, for example, the driver manufacturer, build version, release date, etc. Mandatory attributes are M2M attributes that are required for the operational process of the system and include: the DD's public key (K_{DD}^+); the UUID of its A3C; and the DD's API.

The A3Cs servers also define a set of Machine to Human (M2H) attributes for the managed IoT devices. For example, in a building containing multiple temperature sensors, it is a lot easier to recognize them by their location rather than by their UUIDs. This way, these M2H attributes are stored on A3Cs as they facilitate its management, providing interfaces and an aggregated view over all their names. This solves, for instance, the hurdle of having to reconfigure the device in case it had to be moved to a new location, as it would only required the update of such information at the respective A3C.

The names of the IoT devices are maintained by a name service which runs on each Gateway that provides access to them (through the associated DH). In the bootstrap process, upon connecting to a DH, the Gateway fetches the attributes of all the DDs and their associated IoT devices, followed by the fetching the remaining attributes from their A3C. The relationship between both sets of attributes is assured by the A3C UUID that is part of the (signed) attributes provided by the DD.

The IoT devices' names on their Gateway name service can be updated by their providers, that is, DD and A3C, by presenting an ownership proof: a signature with K_{DD}^- or K_{A3C}^- , respectively. Furthermore, the IoT devices names are also transient, that is, they disappear if the Gateway is turned off or if the DH that provided them disconnects. This way, the naming system is well adapted to respond promptly to any occurring topology changes.

5.3.4 Ticket fetching protocol

The ticket fetching protocol (see Figure 5.4) is a fast, two message protocol which allows an entity, known here as the ticket owner, to obtain a ticket to be used in order to access another entity, considered the ticket target. This ticket is issued by the A3C server of its corresponding target entity.

The protocol starts by the computation of a random value, R_1 , by the owner, which will then be encrypted using the public key of the target A3C (K_{A3C}^+) and sent to the latter along with the owner public key (K_O^+) and the UUID of the ticket target. The A3C, upon receiving this request, first checks if the requester with $UUID_O$ is allowed to access the intended target with $UUID_T$. If so, it will then recover the random value (R_1) using its private key (K_{A3C}^-). Next, it will compute its own random value, R_2 and generate a key by XORing the received value with the new one. Having computed the key, an authorization ticket will be generated

according to the structure previously mentioned in Section 5.3.1 which will be then sent towards the owner, along with the A3C public key (K_{A3C}^+) and its nonce, R_2 , encrypted with the owner public key K_O^+ . Lastly, upon receiving the A3C reply, the owner will proceed to perform the decryption of the received value R_2 using its private key, K_O^- and compute the shared key, which will later act as proof of ticket ownership, when creating a secure session using the fetched ticket.

The protocol, however, does not validate the identity of the requester, nor does it checks if it is a replay request, since it assumes by default that all received requests are genuine. Either way, this is not considered an issue since the reply has no use for rogue requesters.

O	Generates a random R_1
$O \rightarrow A3C$	K_O^+ , $UUID_T$, $\{R_1\}_{K_{A3C}^+}$
A3C	Checks if $UUID_O$ can access $UUID_T$, recovers R_1 with K_{A3C}^- , generates a random R_2 , computes $K = R_1 \oplus R_2$ and generates $T_{A3C}[O \rightarrow T, K]$ with K_T^+
$O \leftarrow A3C$	$\{R_2\}_{K_O^+}$, $T_{A3C}[O \rightarrow T, K]$, K_{A3C}^+
O	Recovers R_2 with K_O^- , computes $K = R_1 \oplus R_2$

Figure 5.4: Ticket fetching protocol for a owner O and a target T. $T_{A3C}[O \rightarrow T, K]$ represents a ticket issued by A3C for allowing O to access T with session key K ; $\{x\}_y$ means x encrypted with key y ; K_Z^- and K_Z^+ are the private and public keys of Z 's asymmetric key pair, respectively.

5.3.5 Session setup protocol

The session creation protocol (see Figure 5.5), is a three-way handshake which, similarly to the ticket fetching protocol, uses a ticket and two random values, R_1 and R_2 , in order to create a derived session key, K' , obtained from the original master session key K , present inside the ticket secret part, already owned by the session requester.

The protocol starts with the session requester, here known as the initiator I, sending towards the target T the authorization ticket (obtained in a previous step according to the ticket fetching protocol), alongside the public key of the target A3C (K_{A3C}^+) and a computed random value, R_1 . The target, upon receiving the request, will first check if the received key K_{A3C}^+ matches the UUID of its A3C server, $UUID_{A3C}$. This verification is performed by digesting the received key and comparing its output with the respective UUID. Following, the target will validate the ticket by checking its signature using K_{A3C}^+ and recovering the session key K from the ticket private part using its private key, K_T^- . If both operations succeed, the target will then proceed to generate a new random value, R_2 , and create a newly derived

I	Generates a random R_1
I \rightarrow T	$T_{A3C} [I \rightarrow T, K], K_{A3C}^+, R_1$
T	Checks if K_{A3C}^+ matches its $UUID_{A3C}$, validates ticket with K_{A3C}^+ , recovers K from the ticket secret part with K_T^- , generates a random R_2 and computes $K' = \text{digest}(K, R_1, R_2)$
I \leftarrow T	$R_2, \{R_1\}_{K'}$
I	Computes K'
I \rightarrow T	$\{R_2\}_{K'}$

Figure 5.5: Session setup protocol between an initiator I and a target T. $T_{A3C} [I \rightarrow T, K]$ represents a ticket issued by A3C for allowing I to access T with session key K ; $\{x\}_y$ means x encrypted with key y ; K_Z^- and K_Z^+ are the private and public keys of Z 's asymmetric key pair, respectively.

session key, K' , from the collective digest of R_1 , R_2 and K . Now, with a key to secure the interactions within the session, the target will reply to I with its value R_2 along with a version of the received value R_1 encrypted with K' . This message is needed so that the initiator may produce the same derived session key K' , while also guaranteeing that the target can verify that it is establishing a session with a valid entity, and not for an attacker that may have stolen the ticket. As such, if the initiator is a legit entity in the network, then it will be able to compute the derived key (since it received the session key, K , from the target A3C in the ticket fetching protocol), and will proceed to confirm it to the target by sending an encrypted version of R_2 using K' . If, otherwise, it is a malicious entity, it will not be able to compute this encryption and the session setup protocol will not succeed. In the end both entities establish a session with secure communications ensured by the derived key K' , unknown to any other entity or eavesdropper.

5.3.6 Client-DD session establishment

Upon the bootstrap of the network and the creation of the required sessions, i.e, the DHM-GW and GW-DH sessions, the Client applications are now able to interact with the desired IoT devices, through their respective DDs. In order to do so, two sessions must first be established: (i) a session between the Client and the Gateway that provides access to the DH supporting the target DD/IoT device and (ii) a session between the Client and the respective DD that controls the IoT device.

In the same fashion as in the network bootstrap configuration, it is also assumed here that a certain discovery protocol occurred in a previous step, thus allowing the network entities to know each others' presence. As such, every Client application knows the location of every Gateway that bridges the communication with the DH/DD/IoT device hierarchy it is allowed

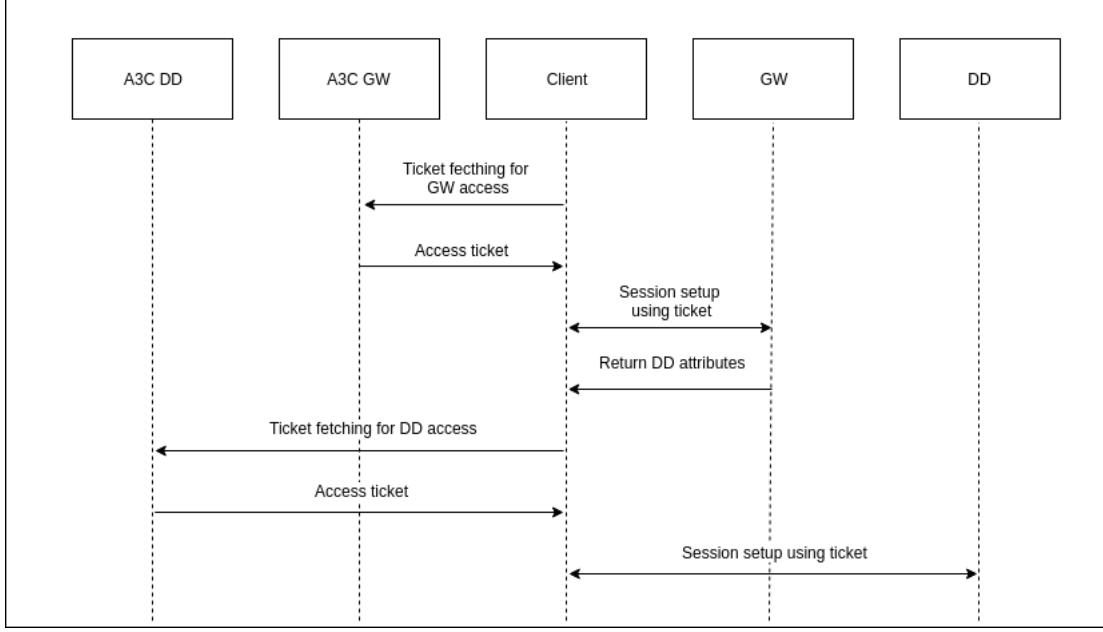


Figure 5.6: Client-DD session establishment (not considering discovery protocols)

to communicate with.

With this in mind, every Client starts the Client-DD session procedure by first fetching a ticket from the respective Gateway A3C and uses such ticket to create a secure session with the Gateway. Upon a valid session creation, the Gateway will then return the attributes of the connected DDs, including its API specification, the public key of the DDs (K_{DH}^+) and the UUIDs of their A3Cs servers. Knowing the DD A3C whereabouts, the client may then issue a ticket request (as specified in Section 5.3.4), and lastly use the fetched ticket to create a valid session with the target DD (see Section 5.3.5). Figure 5.6 shows a visual depiction of the full process to create a Client-DD session.

Prototype implementation

In this chapter we describe the implementation of a prototype of the IoT security architecture, along with an overview of the used technologies, frameworks and software libraries. Furthermore, the protocols explained in the previous chapter are also here presented from a practical standpoint, considering the technologies' features and their inherent constraints.

6.1 THE PHYSICAL PROTOTYPE AND IMPLEMENTED TECHNOLOGIES

For the implementation of the proposed architecture, and given the time restrictions, only one of each of its entities was developed for the final prototype. As such, the deployment of its entities was performed as follows:

- Device Host (DH): Deployed in a Wemos LOLIND32 microcontroller;
- Gateway: Deployed in Raspberry Pi 3 (Model B+) in the local network;
- A3Cs and DHM: Deployed and running in a laptop as local webserver with different logical endpoints
- Client: Deployed as a simple python program, running locally on a laptop.

As for the communication technologies, it was decided that BLE would be used for the non-IP linkage between DHs and Gateways, since both the Raspberry Pi and the microcontroller natively support it. The rest of the communications among the remaining entities are regular IP communications, whether via Ethernet or Wi-Fi in a local network.

Knowing the communication technologies used in the prototype, the remaining of the current section will overview the principal frameworks and libraries applied in its development.

6.1.1 Web server framework - Tornado

Tornado is an open source Python web application framework that takes advantage of non-blocking network I/O, allowing it to scale to thousands of parallel open connections at a time, being thus ideal for IoT networks [133]. This way, it was chosen to be the framework for HTTP-based communication between IP-entities in the local network (A3Cs, DHMs and

GWs). Such communication is performed through a well defined REST API, already known by the communicating parties.

Additionally, Tornado also has built-in support for third-party authentication and authorization schemes, such as OpenID and OAuth, which could be used by Clients for further integration and compatibility with their services.

6.1.2 Persistence frameworks/libraries

6.1.2.1 MongoDB / PyMongo

Each network server, that is, DHMs and A3Cs, contain a local MongoDB database repository, where they store cryptographic information, such as their bootstrap password (properly hashed and salted), their asymmetric key pair and the public keys of the devices they supervise. The MongoDB module is formed by a single database, composed by two collection records, one where the password is stored and another containing the server asymmetric key pair and the public keys of the managed devices. The server communicates with the local MongoDB database through the Python distribution PyMongo, which provides a set of tools for the database management. The PyMongo client (the python instance connecting to the MongoDB repository) for each server can be accessed through the URI *mongodb://localhost:27017/*.

The bootstrap password, that is, the password required for the startup of each server of the network, is securely stored using Password-Based Key Derivation Function 2 (PBKDF2), a specialized algorithm for secure password storage, purposely designed to withstand brute force password cracking, comparatively to other hashing algorithms [1]. This password is also the one used for securely encrypting/decrypting the servers private PEM encoded key file which is performed following the syntax norms of the PKCS #8 standard [62], widely applied for private key information storage.

6.1.2.2 FFat

DHs also need a way to store data in non-volatile memory (namely their asymmetric key pair), so a persistence module of some sort was also required. As such, a File Allocation Table (FAT) flash memory partition was created using the Arduino ESP32 library FFat [41]. This library allows for the creation of a working filesystem in the microcontroller, which would then be used to store two PEM files, one for each DH key (private and public).

6.1.3 MQTT broker - Mosquitto

Mosquitto [30] is a lightweight MQTT message broker, suitable for low power devices and widely adopted in IoT projects. Here it was implemented in the GWs in order to provide an alternative, and more robust mechanism, of handling the DDs responses to the Clients requests.

In order to connect to the Mosquitto broker, the Eclipse Paho [31] MQTT library was used, as it provides a client class enabling the applications to publish, subscribe and process published messages in a very straightforward approach.

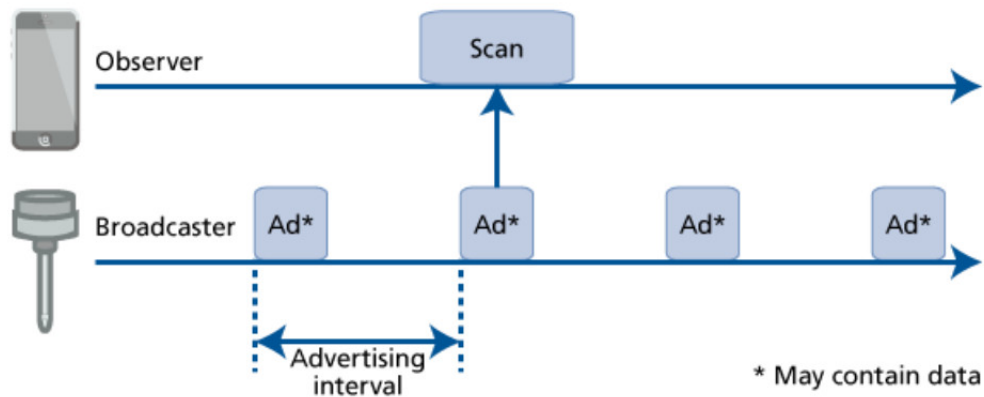


Figure 6.1: BLE advertisement [11]

6.1.4 Bluetooth Low Energy (BLE)

6.1.4.1 Overview of the BLE communication paradigm

As already mentioned, BLE was the protocol of choice for connecting the host device holding the IoT device, the DH, with its endpoints, the GWs. As such, before understanding the DH workflow itself, it is important to grasp the details behind the establishment of a BLE connection.

As also previously seen in Section 2.7.4.1, the BLE communication is performed via device advertisement, previous to the establishment of a point-to-point connection in a client/server approach. During this advertisement phase, the advertiser entity is known as the "Broadcaster", while the client entity, known as the "Observer" in this scenario, is scanning for new BLE devices in its vicinity (see Figure 6.1). Upon finding a wanted device, the client may then initiate a connection. When such link is established, the client controls the communication, either through the direct transmission of data or by polling the advertising device, here known as the "Server" (see Figure 6.2) [11].

However, in order to properly follow the BLE communication, it is necessary to understand its underlying architecture. An overview of such architecture can be seen in the next section.

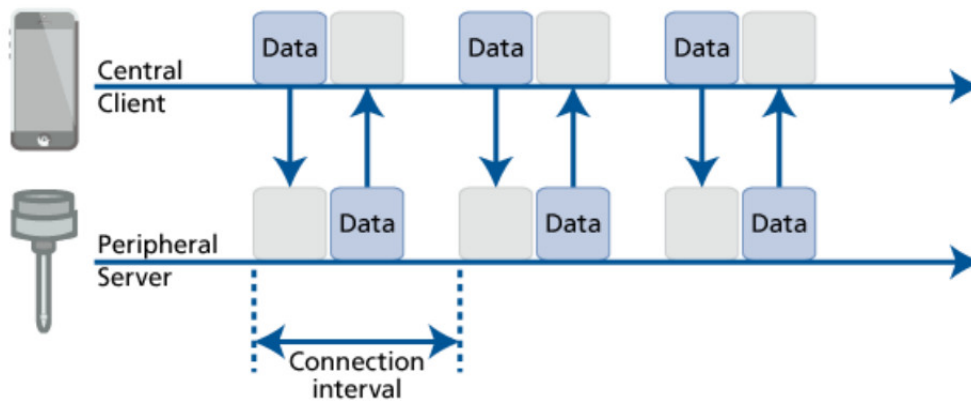


Figure 6.2: BLE connection [11]

6.1.4.2 BLE Architecture

Without delving into much detail and specificities, the BLE architecture is composed by three major layers, the Application layer, the Host layer and the Controller layer, the last two being themselves constituted by other additional sub-layers (see Figure 6.3). Such layers are described as follows [21]:

- Application layer: As the name suggests, this is the layer where the general application interface, which makes use of BLE technology, is defined.
- Host layer: The host layer is itself subdivided into five distinct sub-layers:
 - Generic Access Profile (GAP): Dictates how the BLE devices interact with each

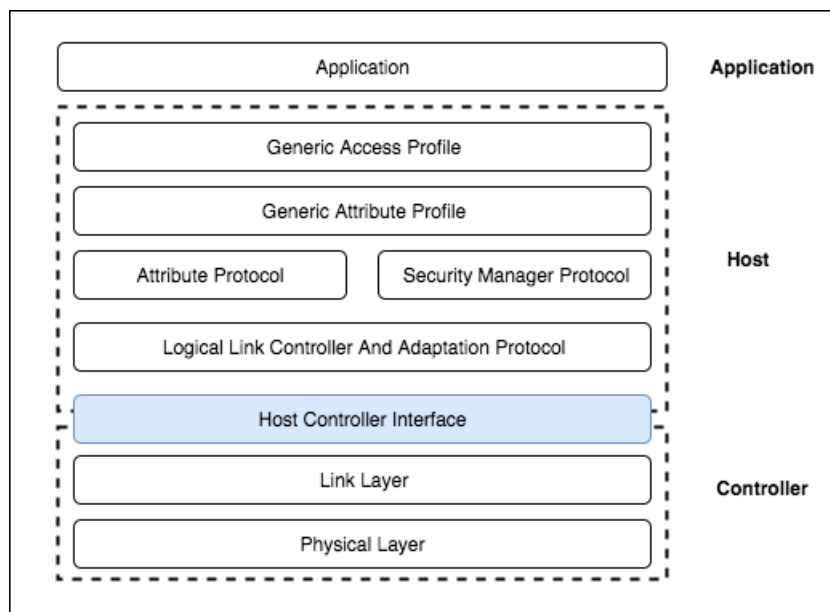


Figure 6.3: BLE architectural layers [21]

other, operational modes and security aspects of the communication, namely security modes and procedures. Here, the devices are always categorized as having one of two roles:

1. Broadcaster: A BLE device that advertises information to its vicinity. The broadcaster is also known as peripheral.
 2. Observer: Opposite to the broadcaster, the observer listens for any advertising packets it may receive, followed by the creation of a connection if desired.
- Generic Attribute Profile (GATT): Provides a detailed definition on how the BLE attributes are structured, encapsulated and transmitted between the devices. These attributes are organized in a hierarchical model where BLE services are defined, each containing its own set of user data, formally known as characteristics (that is, containers for user data). Similarly to the GAP, the GATT protocol also defines roles for the connected devices:
 1. Client: the entity that sends requests and receives updates from a server with an already established connection. The information transaction is based on read/write operations performed over the server characteristics.
 2. Server: entity responsible for storing the data, organized in the form of attributes, to be read/written by clients. It may also send unidirectional updates to the clients if such is configured.
 - Attribute protocol (ATT): Despite the client/server roles being enforced by the GATT protocol, these are defined by the ATT protocol which implements the rules for accessing the attributes/data the device contains (none, readable, writable, readable/writable). Such data, as already mentioned, is arranged in the form of attributes. An attribute is defined by a UUID, a set of permissions over the attribute and the value itself.

Additionally, the ATT protocol also defines the set of operations over the attributes:

 1. Read: the Client obtains an attribute from the Server.
 2. Write: the Client updates the Server attribute value.
 3. Notification: the Server voluntarily and asynchronously sends data to the Client. The Client, however, must first allow such notifications.
 4. Indication: similarly to the notification operation, except here the Server expects an acknowledgement response from the Client.
 - Security manager protocol: defines authentication between BLE devices.
 - Logical Link Controller and Adaptation Protocol L2CAP: Responsible for the fragmentation/defragmentation of transmitted data, as well as the multiplexing/demultiplexing of the communication channels over the mutual link.
- Controller layer: corresponds to the BLE hardware responsible for the RF communication.

6.1.4.3 BLE Frameworks/Libraries

Regarding the libraries used for BLE exploration, the GW software, developed in Python, makes use of the open source library bluepy [50] which provides a high-level API for the establishment of BLE communications. On the other hand, the DH uses the ESP32 BLE library [69], a user friendly C++ wrapper over the more detailed and complex ESP-IDF BLE library. In Section 6.2.1, the BLE model here presented will be further detailed in the context of the prototype developed and its inherent BLE communications.

6.1.5 Data serialization - Protocol Buffers

As we saw, the DH must be addressed as a constrained computational environment (even though the used microcontroller Wemos LOLIND32 is far more performant than other legacy devices). As such, there is a clear requirement to minimize its throughput, i.e the number of exchanged messages with its Gateway, while also keeping the size of exchanged messages to a minimum. Furthermore, there was also the need to properly serialize the messages transmitted between the DHs and GWs, entities which were developed using different programming languages, in such a way that interoperability is assured, resulting in no discrepancies in the interpretation of the received data at each end.

This way, Protocol Buffers [42] were used for performing fast and simple serialization/deserialization of structured data. It works by defining messages, that is, data structures specified in protocol definition files, which are then compiled using the *protoc* compiler, resulting in code for serializing and interpreting such data in the considered programming language (Python and C++).

In the prototype implementation, both the GWs and the DHs share the same protocol definition files and their resulting classes after compilation. Such code is then imported and used for encoding/decoding the transmitted messages. The protocol file created for defining the messages exchanged through BLE can be seen in Appendix A.

For the DH, the Nanopb [3] library was used. It is a small implementation of Protocol Buffers in ANSI C, targeted for memory restricted systems, such as microcontrollers.

6.1.6 Cryptography

Regarding cryptography, each entity in the architecture contains its own cryptographic library, containing the symmetric, asymmetric and digest algorithms used along the security operations of the architecture. For the IP-connected network entities (A3Cs, DHMs and GWs), the Cryptography.io [139] library was chosen and explored, both for its rich documentation and high-level cryptographic interface. As for the DH, the cryptographic library mbedTLS [123] was used, as it takes advantage of the cryptographic hardware acceleration module of the used microcontroller (Wemos LOLIND32).

6.1.6.1 Cryptographic algorithms and primitives

- 2048 bit RSA key pairs (as recommended by NIST [20]).
- PKCS #1v1.5 padding for easier interoperability in the asymmetric encryption operations between DH (MbedTLS) and other entities (Cryptography.io).

- Optimal Asymmetric Encryption Padding (OAEP) and Probabilistic Signature Scheme (PSS) for handling signatures between IP-connected entities.
- Advanced Encryption Standard (AES) with 256-bit keys for all symmetric encryption operations (using CBC mode).
- Hash-based Message Authentication Code (HMAC) using Secure Hash Algorithm (SHA)-256 for securing sessions between entities.
- MD5 for 128-bit UUID generation.

		Module/Service			
		Cryptography	BLE	Data serialization	Persistence
Entity	DH	MbedTLS	ESP32 BLE	Nanopb	FFat
	A3C	Cryptography.io			PyMongo
	DHM	Cryptography.io		Protoc/Protobuf	PyMongo
	GW	Cryptography.io	Bluepy	Protoc/Protobuf	

Table 6.1: Programming libraries, used for each module and entity

6.2 ENTITIES' LIFE CYCLE AND PROCESSING WORKFLOWS

6.2.1 Device Host (DH)

When the DH starts, the first step in its life cycle is to mount the FAT file system partition, which contains the public key pair of the DH and the public key of his corresponding supervisor entity, the DHM. This does not mean, however, that such keys are necessarily pre-installed in the device itself, as it occurs in many deployments of IoT devices (considering their default factory configurations for example). This way, we may also consider the possibility that these same keys could also be distributed by the devices DHMs, through a proper discovery and configuration protocol executed at the IoT system startup, followed by their local storage. Such protocol, was, however, not considered for this dissertation. This particular key distribution issue is further discussed in the next chapter, regarding the architecture security evaluation.

With all the cryptographic information properly loaded, the next step is to initiate the BLE module by configuring a BLE server. Such configuration includes the assignment of a UUID to the BLE server itself (independent from the DH UUID) as well as the specification of its underlying characteristics and their corresponding UUIDs. These characteristics, as already seen in Section 6.1.4.2, are containers that hold data to be transferred across the BLE link. A schematic of such structure can be seen in Figure 6.4.

The BLE server defines only a singular characteristic, which is used for holding the data in the communications with the Gateways. As seen, the DH only performs its duties, that is, processing client requests and generating their respective responses after receiving the authentication ticket from a certain GW (a ticket generated by its DHM). As such, the DH, after finishing the configuration process, that is, loading cryptographic data from memory and initializing the BLE server, will then start to announce its presence in the network to any nearby

devices. This advertising is performed by writing the DH internal state into the communication characteristic, followed by the BLE broadcasting of such characteristic. The internal state of the DH, upon successful startup and configuration, is always `DH_AUTH_REQ` and during this period the DH is suspended from performing any other actions as it waits to receive a session request from a Gateway. All of the considered internal states of the DH can be seen in Appendix H (note that most states are respective to error processing and validation, which were not considered as a priority in the developed architecture, thus not implemented in the workflow of the current solution).

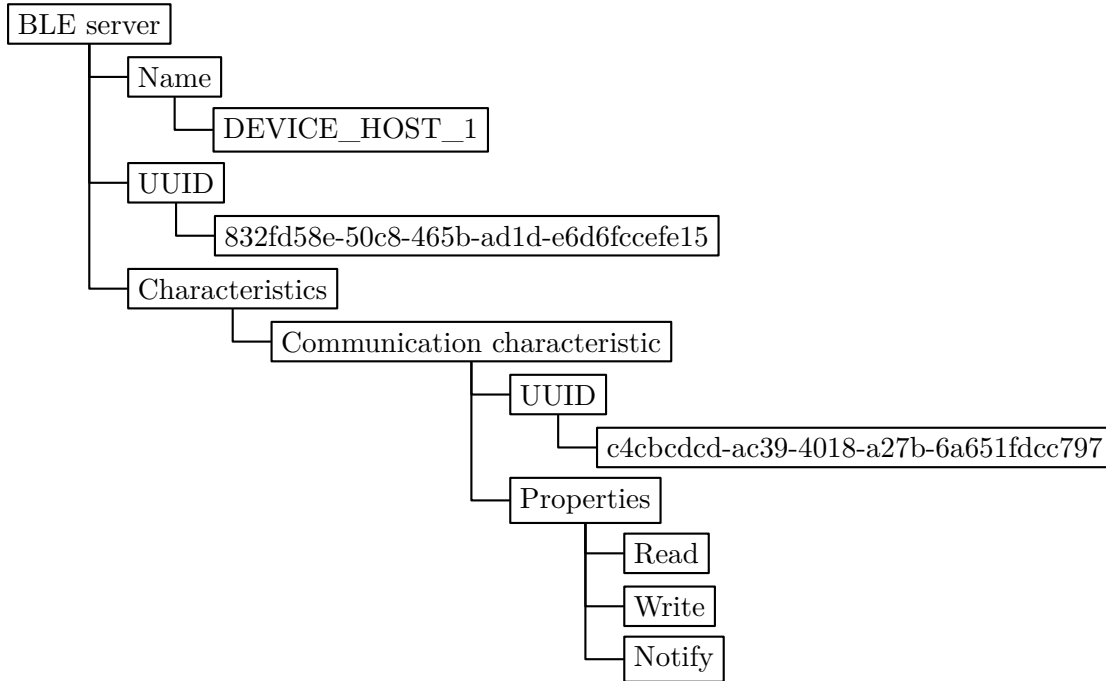


Figure 6.4: Internal structure of the DH BLE server.

Upon the reception of an access ticket, the DH will then proceed to validate its signature and follow the rest of the steps of the session setup protocol as detailed in Section 5.3.5. DH responses towards the GW in this link are done in the form of BLE notifications.

Upon establishing a session with a Gateway of the IoT network, the DH communicates to the GW the information about the installed DDs, namely their API, its public key and the public key of their A3C server. This is done so that the GW, which is now a trusted entity, may communicate its DDs information to Clients applications that aim to interact with them.

With this bootstrap process properly executed, the DH is now ready to process client requests and generate their corresponding responses. All messages exchanged within a session are always properly secured using the shared session key, which is used either for computing HMACs to ensure the confidentiality of the message and to encrypt the message. Such encryption is not enabled by default, but may be configured as a tunable parameter that may be set up during the processing of the access request to the DD by the Client.

6.2.2 Gateway (GW)

In order for GWs to start operating, they require a bootstrap password which must be provided via human input. Such password is also the one used to encrypt the file of the Gateway private key file. Following this step, the HTTP server is initiated and the GW stays idle, waiting for any upcoming connection. Every message exchanged between IP-connected entities with the gateway follows its REST API, which can be seen in Table 6.2.

As seen in Section 5.3.2, during the bootstrap setup the DHM will be the first entity to communicate with the GWs, establishing a mutual secure session followed by the upload of configuration data, including tickets, used to access the slave DHs. This ticket contains in its public part the DHs BLE address, required for the GW to discover and connect to the devices it is instructed to.

Upon receiving such configuration data, the GW will scan for nearby, BLE compatible devices (acting as a BLE observer) and try to initiate a connection with the DHs matching the received BLE addresses. That connection is established by reading the value of the DH communication characteristic, which should indicate a valid internal state of the device, in this case `DH_AUTH_REQ`. If that's the case then the GW can proceed to start the session with the DH by sending the received ticket, along with the DH A3C public key and a nonce value, following the session setup protocol already presented in Section 5.3.5.

If a target device is, however, either not found at the time a GW receives its configuration data, or if the read characteristic value indicates an invalid internal, the GW will mark the device as unavailable and will try to start a new session attempt after a specified time interval (as long the ticket is still valid).

After the correct session establishment, the DH will then upload the information of its DDs to the now trusted GW, which is now ready to process any incoming Client requests.

Method	URL	Params	Param values
POST	/dhmSessionSetup/	ticket public key nonce1	bytes bytes bytes
POST	/dhmSessionSetup/validation/	encrypted_nonce2	bytes
POST	/dhTickets/	data signature	bytes string
POST	/clientSessionSetup/	ticket public key nonce1	bytes bytes bytes
POST	/clientSessionSetup/validation/	encrypted_nonce2	bytes
POST	/clientDDSessionSetup/	data signature	bytes string
POST	/clientDDSessionSetup/validation/	encrypted_nonce2	bytes
POST	/clientRequest/	request_type target_dd action	string string string

Table 6.2: Gateway REST API specification.

The GW runs an MQTT broker module, through which Clients may choose to receive the responses to their DD requests. Upon reception of the DD response, the GW publishes it on the fly to the corresponding topic. Such topics follow the syntax `/ble/<DD_UUID>/<DD_API_FUNCTION>/`, where `DD_UUID` corresponds to the UUID of the target DD and `DD_API_FUNCTION` to one of the functions of the DD API, which will be addressed in Section 6.2.6.

6.2.3 Device Host Manager (DHM)

Similarly to GWs, the DHM also requires an initialization password in order to start and load its keys. However, here the DHM will firstly initiate the configuration of the IoT network by establishing the necessary sessions with GWs; only after does it starts its web server to process other events such as, for example, the handling of expired sessions.

To begin with, the DHM will establish sessions with the GWs that will act as endpoints for the DHs it manages. For that, the DHM will first contact each GW A3C in order to fetch the required access tickets in order to properly access the target GWs. Second, after session establishment, it generates access tickets for DHs (with configuration properties), in order for the GWs to discover their slave DHs. After successful session establishment with the GWs, the DHM may receive a confirmation message from each GW of such outcome.

6.2.4 A3C server

As with the rest of the IP addressable network entities, the A3C also starts with a given password, followed by the loading of its key pair PEM files. Next it will start its web server and listen to any ticket fetching requests following the rules of the ticket fetching protocol as specified in Section 5.3.4. The REST API of a A3C server can be seen in Table 6.3.

Note, however, that in the developed solution, the configuration and installation of missing DDs performed by the DD A3C in the network bootstrap phase (see Section 5.3.2) was not implemented.

Method	URL	Params	Param values
POST	/ticketFetch/	target_uuid public key encrypted_nonce_1	string bytes bytes

Table 6.3: A3C REST API specification.

6.2.5 IoT devices

For the IoT devices in the physical prototype, for testing purposes, they are represented as both the ESP32 internal temperature sensor, as a generic IoT sensor, and the ESP32 embedded LED, acting as a generic actuator device. In a real environment, however, the device would be a real sensor natively decoupled from the microcontroller and connected to it via a serial bus (SPI, I2C, etc.).

6.2.6 Device Driver (DD)

Regarding the DDs of the prototype implementation, they consist of the required code for interacting with the temperature sensor and the led of the microcontroller. Additionally to the standard code, it also includes its respective asymmetric key pair, the UUID of its A3C and a description of its API, which is communicated to the GW in order to inform the clients of the presence of the IoT device in the network. The API specification of these DDs can be seen in table 6.4

Driver	API function	Return type
Temperature	readValue() calibrateSensor() setUnits(char temp_units)	uint8_t int int
Light	turnOff() turnOn() getStatus() setBrightness(int value) increaseBrightness() decreaseBrightness()	void void boolean int void void

Table 6.4: DDs API specification.

6.2.7 Client

The Client is a simple python application that interacts with the IoT devices, through the previously described DDs APIs. As with the other architectural entities, it starts by loading its asymmetric key pair. Following this, it fetches a ticket from the A3C of the target GW and uses it to establish a secure session with it. This GW acts as the endpoint of the DD the Client wishes to access. Upon successfully establishing the session, the Client will download the DD data from the GW, namely the available API functions, its public key and the UUID of its A3C server. With this information the Client can then discover the location of the DD A3C in the local network and fetch from it an authorization ticket (assuming a discovery protocol is implemented). Having received the ticket, the Client then starts a secure session with the target DD by sending associated messages towards the GW, which will then relay them to the DH running the DD. Once the session with the target DD is established, the Client may then interact with IoT device using its DD API.

For the developed prototype, the Client establishes a session with the temperature driver and issues a readValue() request in order to obtain a single reading from the temperature sensor. The GW will then relay the value to Client, through the already established Client ↔ GW secure session.

Since the GW runs an MQTT broker, the Client may receive the responses to the issued requests by subscribing to its associated topics and wait until such responses are published by the GW upon their reception from the respective DDs. The MQTT topics for Client ↔ DD interaction obey the syntax /ble/<DD_UUID>/<DD_API_FUNCTION>/, where

DD_UUID corresponds to the UUID of the target DD and DD_API_FUNCTION to one of functions of the DD API, as the specified in Table 6.4.

In the developed solution, the Client first subscribes to the topic /ble/fl7a1fe9d42b711c463cdad54e5db6f0/readValue and then sends the requests towards the DD connected GW. The GW will then publish the received response on the same topic. The Client then receives in the topic the byte value 123, which corresponds to the internal temperature of the ESP32 device in Fahrenheit degrees. The debug logs of every architectural entity, from the bootstrap of the network to performing this action, can be seen in Appendixes B, C, D, E, F and G.

Security evaluation and results

Measuring the security of a software project is not a trivial process, mainly since there is not a clear set of standard guidelines or metrics that developers must follow in order to secure their specific solutions, joined with the fact that it is nearly impossible to predict every scenario where the system may have security flaws that could be abused by attackers. Even if a developer could guarantee that every module of its project is secure, there would still be vulnerabilities induced by human error, such as the classic bad practices of passwords written in clear text files or post-it notes. Nonetheless, as already seen in Chapter 3, there are some properties that must be present in the solution in order to improve its security, as well as a set of commonly present vulnerabilities that must be identified and properly handled.

This way, the current chapter aims to evaluate the designed system from a security perspective, comparing its performance towards the more pertinent security vulnerabilities affecting the IoT architectural model as well as the developed prototype, which introduces additional vulnerabilities given its adopted technologies, namely with the use of the BLE protocol.

7.1 SECURITY EVALUATION OF THE THEORETICAL MODEL AND ITS UNDERLYING PROTOCOLS

Regarding privacy concerns, the developed solution defines a clear segregation between the local IP network and the rest of the non-IP addressable IoT devices, through the implementation of authorized Gateways as bridges for both domains. This architectural definition prevents IoT devices (through their DDs) from sending data to unknown arbitrary locations other than the authorized Clients, through the properly authorized Gateway.

Additionally, this property also prevents IoT devices from leaking any identity-related information about who accessed their information or controlled their activity, mostly since the authorization tickets do not contain any sort of identification of its associated owner.

One major advantage of the proposal is the clear division of roles regarding the definition and enforcement of access control policies for accessing IoT devices. While the access policies

are defined at the A3C servers and DHMs, their actual enforcement is offloaded to the other architectural components that provide connectivity and access to the IoT devices (through their DDs), the GWs and the DHs. This property allows the definition of fine-grained access control policies to the IoT devices and their produced data, ultimately granting more flexibility than many IoT systems, which usually either allow or restrict the access to the device as a whole. For example, the security permissions for accessing an actuator of a home oven should be much more strictly designed and enforced than those for turning a light bulb on and off. While the misuse of the first example could cause a house fire, threatening human lives, the second would only result in a broken light bulb. By using our solution, this problem would be tackled through the definition of risk-based access control permissions, which would be applied to each device.

Furthermore, this decentralization into A3Cs also grants more scalability and flexibility to be adjusted for a wide range of different scenarios, such as having only a single A3C for a very small network or up to several for each component that requires an A3C server. In the same note, one additional advantage comes from the implicit distributed nature of the system which grants it more fault tolerance, while being able to bootstrap itself from a simple initial configuration.

Additionally, regarding the applied cryptography, the adoptance of an asymmetric scheme for authenticating the network entities based on pre-known public keys (or the related UUID) also provides performance benefits when compared with the usual protocols that require a PKI or the use of certificates. Nonetheless, it is worth noting that, as mentioned in the previous chapters, the use of asymmetric cryptography in IoT devices is far from ideal, as it is computationally expensive and possibly not supported by constrained devices. Despite this issue, in the developed protocols the use of asymmetric cryptography is not frequent and kept to a minimum, mostly used when establishing a secure session between entities. Once the session is established, and throughout its duration, all its underlying security is performed using the more efficient symmetric schemes. The implementation of elliptic curve based algorithms for the DD/DH cryptographic procedures could lighten this problem, as they are proven to be more efficient in constrained devices than its classic RSA counterpart [98]. This way, it seems clear that the security benefits provided by this scheme largely outweighs the extra processing power required by the device for session setup.

Besides performance, given the exclusion of PKI and certificate-based schemes in the solution, the non-repudiation property is not achievable and it is one of the drawbacks to be considered.

7.1.1 Device spoofing and impersonation

This section describes how an attacker could impersonate or spoof the devices in the IoT network, namely the DHs and GWs, and how the system would react to such an occurrence.

7.1.1.1 Device Host (DH)

The impersonation of a valid DH in the network is not an easy task, mainly since each DH contains a unique asymmetric key pair that singularly identifies it as a legitimate entity in the

network. For example, in a scenario where an attacker were to spoof the BLE address of a legitimate device and force it to establish a session with a GW, this attempt would inevitably fail when extracting the session key of the ticket received from the GW (generated by a DHM). Such operation requires the knowledge of the private key of the device being spoofed and impersonated, which is only possible through the extraction of the key pair from the DH. The secure storage of the DHs private keys will be further addressed in Section 7.2.

In order to properly insert a malicious DH in the network, the attacker would have to access its corresponding DHM and manage to add it to the list of valid DHs it manages.

7.1.1.2 Gateway (GW)

The GW, in order to connect with a DH, must first establish a session with its DHM. This session is established based on the ticket the DHM fetched from the GWs A3C server. As such, and in the same fashion as with the DH case, if an attacker intended to impersonate a GW, this attempt would fail since the GW contains an asymmetric key pair that uniquely identifies it in the network, and would require their disclosure or extraction from the GW.

Similarly to the DH case, in order to insert a new malicious gateway in the network, it would have to be manually added by the attacker to the list of legitimate Gateways at the respective GWs A3C database.

7.1.2 Ticket stealing / Man-in-the-middle (MITM)

Probably one of the main occurring issues when handling ticket-based authentication is that an issued ticket could be stolen by an attacker eavesdropping a network connection. By doing so, the attacker could then try to use the ticket to establish a malicious session with its target, undetected by the latter.

In the ticket fetching protocol, which is followed by the session setup protocol, this problem does not occur. For example, consider Mallory an attacker in the IoT network that managed to eavesdrop the communication between a legitimate Client and a GW A3C server, during the session establishment for accessing a certain DD. By doing so he managed to steal the ticket provided to the Client by the GW A3C during the ticket fetching protocol. By using such ticket, he could attempt to impersonate the original Client, that issued the ticket request, and thus attempt to create a rogue session with the GW. However, such effort would not succeed, particularly due to last message exchange of the session setup protocol, which is done especially to counter this particular attack. When issuing a ticket request to the A3C, the legitimate client received in the last message of the ticket fetching protocol the second nonce (R_2) for session key calculation, which was properly encrypted using the Client public key (and thus can only be decrypted by the Client itself using its private key). This way, a rogue Client, despite having stolen the session ticket, has no knowledge of the session key, thus is not able to calculate the derived session key, which requires the previous session key and the two exchanged nonces. Therefore, the malicious Client never manages to properly generate the last message of the session setup protocol, thus is not able to create a legitimate session using a stolen ticket.

7.1.3 Privacy and enumeration

As already seen, the developed model is not affected by any major privacy or enumeration vulnerabilities, mostly due to the fact that only properly authorized entities can establish secure sessions with the online devices, followed by the retrieval of the desired attributes or produced data.

Nonetheless, although the architectural model itself is not affected by any major privacy or enumeration vulnerabilities, these are present in the developed prototype due to the adoptance of BLE as the underlying communication technology. In this security context, enumeration is considered to be the process through which attackers may extract machine names, network resources or other information from a given system. These issues will be particularly addressed in the next section under the BLE security overview (Section 7.2.1).

7.2 SECURITY EVALUATION REGARDING THE PROTOTYPE AND ADOPTED TECHNOLOGIES

Regarding the IP connected devices, each of their RSA key pair is securely stored using the password provided to them during startup. Such password is also securely stored in the database using the specialized algorithm PBKDF2. It takes as input the password, a salt value, an iteration number, the length of the output key and a hashing algorithm. Then it applies the hashing algorithm over the password and salt as many times as the specified iteration number, resulting in a derived key which is then stored, instead of the original password.

However, in the DHs, and in the DDs they run, such storage mechanism is not present since the DHs do not have any sort of physical input through which a user could introduce a password to be used to encrypt its private key. Additionally, the secure provisioning of the DH with its key pair, is a challenging issue and not here considered. On the other hand, one could argue that the private key could be stored in the device using a password, only known by its DHM, provided to the DH during the startup phase. Nonetheless, the same problem is here present since, in order to send the password value in a secure manner, the DHM would have to encrypt it using the DHs public key, followed by its decryption upon reception. This way, we face a deadlock-like scenario: the received encrypted password is meant to be used to decrypt and load the DHs private key, which in turn it must have in order to decrypt the password value.

This way, key extraction is one of the major issues concerning the developed prototype, since an attacker could gain physical access to the device and extract the plaintext stored keys with ease. One possible solution to overcome this problem would be through the integration of a TPM [58] on each DH, allowing the secure storage of the keys and prevent their extraction.

The ESP32 microcontroller contains some security provisions that could be explored, such as secure boot and flash encryption [122]. The secure boot objective is to ensure that only signed and trusted software uploaded to the device may be executed, this way countering any tampering attempt by a malicious entity. As for the flash encryption, it is a mechanism that ensures the encryption of the application firmware stored in the flash memory of the device.

This one is usually used by manufacturers for securely shipping their proprietary firmware in the devices.

7.2.1 BLE security overview

Although not specifically used in the development of the prototype (since all of the applied security is performed at the applicational level), BLE contains some security definitions applied to the underlying data in the connection, while also introducing some implicit vulnerabilities which must be addressed in order to grasp how secure it is as a lightweight technology for connecting IoT devices. As such, these vulnerabilities must be considered in order to fully evaluate the security of the developed prototype.

The major affecting BLE are considered to be eavesdropping, man-in-the-middle (MITM) and identity tracking [23].

Regarding eavesdropping, BLE tries to overcome this problem through the encryption of the exchanged data (using AES in Counter mode with CBC-MAC (CCM)). However, this effort is not enough since, in order to encrypt the data, both communication devices need to first establish a shared secret key (through one of four pairing methods which will be explained next), becoming vulnerable in case of passive listening by an attacker.

In the same fashion, through a MITM attack, an attacker could impersonate a legitimate device into establishing a compromised connection. As with eavesdropping, the resilience against MITM attacks depends on the pairing method defined for the BLE communication.

Lastly, identity tracking refers to the capability of an attacker associating a BLE device address to a certain entity (person or object) while physically tracking it based on the presence of the device. The BLE approach for solving this issue is through the periodical change of devices' addresses [23].

Regarding pairing, it is the process through which the BLE devices create a secure connection through the establishment of a shared key, known as Temporary Key (TK), which is then used to create a Short Term Key (STK), specifically used to encrypt the connection. In the most recent devices (Bluetooth v4.2 devices), however, Elliptic Curve Diffie Hellman (ECDH) is used for generating a Long Term Key (LTK), not using either TK and STK. This way, pairing can be divided into four different types, each regarding the type of Input/Output (I/O) capabilities of the devices, resulting in different levels of security [121]:

- **Numeric Comparison:** Uses ECDH (thus only available for Bluetooth v4.2 compatible devices) for generating a 6 digit number followed by human input confirmation in order to derive the STK. First both the initiator device and the responding device present their I/O capabilities to each other, followed by the exchange of their public keys. Both devices then exchange a nonce value, from which they are able to compute the matching 6 digit number. This number is then displayed in the devices screens or LCD displays and the user needs to make sure they match, usually by pressing a button (deemed the OK button). By receiving confirmation on both devices, they are then able to derive the STK. The visual confirmation is required in order to enhance the protection against eavesdropping and MITM attacks.

- **Out of Band (OOB):** Designed for scenarios that use a wireless communication mechanism, such as NFC, for discovering devices and exchanging cryptographic information used in the BLE pairing process. First both the initiator device and the responding device present their I/O capabilities to each other. Next both devices calculate a random value and exchange TKs using the OOB link. Having the keys, they then initiate a validation step using the random values in order to finalize the pairing process. The main advantages of this method is that it allows for larger TKs, up to 128-bits, enhancing the connection security. Additionally, if the OOB channel is protected from eavesdropping and MITM attacks, then it can be assumed that the BLE connection will also be protected against those vulnerabilities. Furthermore, it is also a more flexible option for developers and they are freely to define their own pairing mechanisms.
- **Passkey:** Used when one of the devices has a display and the other an input interface such as a keyboard. First, the initiating device generates a random 6 digit number, the TK, which will then be shown in its display. The receiver device then requests the user read the TK from the display and write in to the device using its keyboard. Upon the TK insertion, both devices will generate a 128-bit random value and initiate a validation step through the exchange of both values. Upon successful validation, they then derive the same Short Term Key (STK) from the previously generated TK and random values. This method is secure as long as an attacker did not eavesdrop the connection during the pairing process and did not sniff the exchanged values. If this can be assured, and if the attacker is not able to obtain the TK via other means than the BLE connection (MITM), then the connection is deemed secure.
- **Just Works:** most common pairing mechanism and also the least secure, used when one or both devices have no I/O capabilities. In legacy devices (Bluetooth v4.0 and v4.1) the TK is always equal to 0 and the STK is generated on both devices through packets exchanged in plain text. In version 4.2 the devices exchange their keys and random values similarly to the numeric comparison. However here there is no confirmation through the display and each device generates a confirmation value to be exchanged in a validation stage. First the responding device generates a random value C_b , to be sent towards the initiating device along with its already generated nonce. Meanwhile the initiating device will do the same procedure and send the values towards the responding device. Upon exchange, the responding device uses the initiating device nonce to generate its own confirmation value C_a , which should match the received C_b value, validating the connection. Although Just Works on Bluetooth 4.2 compatible devices is more resilient against eavesdropping than its counterpart, it still suffers against MITM attacks as there is no visual confirmation, unlike in Numeric Comparison.

Since the GWs and DHs of the architectural model do not have any sort of specific user interface, such as keyboard or display, the only pairing method of interest would be the Just Works. This mechanism, however, unlike the other pairing methods, is the least secure of them all, since it is vulnerable against MITM attacks. Still, it is important to note that despite the other methods being more secure than Just Works, they too are vulnerable to

attacks. For instance, in [112] the author presents a method to bypass Passkey authentication without making use of passive eavesdropping to discover the shared secrets.

This way, since all the security applied to the communications of the prototype is performed at the applicational level, the first two presented BLE vulnerabilities, eavesdropping and MITM, are not of great concern. However, identity tracking is still present, since the BLE address of DH devices is of static nature, especially when considering that the GWs learn about the DHs by the address provided by their DHM. Nonetheless, this does not exclude the possibility of using address randomization, which would require some sort of additional protocol in order to synchronize the attribution of new addresses both at the DH and DHM, while also guaranteeing their uniqueness in the network.

Lastly, one of the BLE inherent security and privacy issues occurs due to its advertising properties (already seen in Section 6.1.4.1), which make it impossible to hide the devices' presence, leading to an entry point for every attacker seeking to explore any vulnerabilities that may exist. To minimize this problem, a BLE device may apply address randomization, to prevent an attacker to monitor its activity over time, or whitelisting, that is, only establishing connections with trusted (listed) devices [35]. The latter, however, can be easily exploited by an attacker, as it just has to personify the address of a GW present in the DHs whitelist.

This way, in the developed prototype, the same problems are present since the DHs will continuously advertise their presence in the network after their startup procedures.

BLE-Guardian [35] is a new device-agnostic system that aims to tackle this issue of protecting privacy in BLE environments. It does so by enabling users and administrators to manage the entities that can discover, scan and connect to the respective BLE devices with resulting small performance drops and overheads. However, it is not applicable in our architecture as its access control module uses Out of Band (OOB) pairing for authorizing devices to scan and interact with other BLE enabled devices.

7.3 PERFORMANCE OF CRYPTOGRAPHIC ALGORITHMS

The most common cryptographic operations in our model are based on symmetric key and occur during the communication of data through a valid session. Cryptography is used for assuring confidentiality, or for computing MAC values for integrity control. As such, studying and testing their performance is not of great concern as symmetric key encryption algorithms are far less resource demanding than its asymmetric key counterparts.

However, what could be an issue of the implementation, and a relevant point of discussion, is the use of RSA as the algorithm for performing asymmetric key cryptography operations, instead of ECC, which are generally considered more suitable for resource constricted scenarios, such as the IoT.

In this section we will present the performance of the applied cryptographic algorithms, considering the different hardware used in the developed solution, and we will compare their performance against some of ECC algorithms.

7.3.1 DH

Implemented in an ESP32 device, the only asymmetric operations the DH performs occur during the session setup protocol:

- One RSA signature validation of the received authorization ticket signature;
- One RSA decryption of the ticket private part, to obtain the ticket session key.

This way, it is possible to argue that the signature validation could be upgraded from the used RSA scheme to one of the considered Elliptic Curve Digital Signature Algorithm (ECDSA) curves. In order to test this case, RSA operations were tested against the curves SECP224R1 (224-bit key), SECP256R1 (256-bit key) and SECP374R1 (384-bit key). As already seen in Section 6.1.6.1, the DH has a 2048-bit RSA key. Table 7.1 presents the average time required (in seconds) for performing each cryptographic operation for each algorithm from a total of 1000 tests. The signature operations were performed over a 412-byte sample, as it is the average length of the ticket data received by the DH, and its validation operations performed over the previous output (a 256-byte signature). As for the decryption operation, a 256-byte sample was used. Since in ECC there is no encryption/decryption operations directly using their keys, these are marked as gray.

Given the obtained results, it is possible to observe that the RSA signature validation is much faster than its ECC counterparts, thus being the appropriate algorithm for this operation. However, while the signature may be efficient, its decryption is not, being the downside of its application as the most heavy operation the DH performs.

Algorithm	Decrypt (s)	Signature validation (s)
RSA	7.74333e-1	2.70860e-2
ECDSA SECP224R1		2.41167e-1
ECDSA SECP256R1		3.37117e-1
ECDSA SECP384R1		4.86750e-1

Table 7.1: ESP32 performance comparison between RSA and different ECDSA curves. Signature validation operations were performed over 256 bytes of data. Decryption operations were performed over 256 bytes of data.

One possible solution to improve this performance would be through the introduction of ECDH to complement the performance drop when decrypting with RSA. With ECDH both communicating parties exchange a public value in order to derive a shared key. This key could then be used to perform decryption operations using a symmetric key algorithm, in our case AES. This introduction would not require additional messages, only some small changes to the ticket fetching and the session setup protocol, as the DHM and DH would have to exchange their public values in order to derive a shared key.

In order to adopt ECDH, prior to any other interactions, the ticket issuers (the A3Cs) would have to establish a shared key with the ticket requesting entities and also a shared key with the entities they supervise (the targets of the requesters). Having the shared keys, the ticket fetching protocol will stay the same, only difference being that now the exchanged nonces are encrypted using the shared key (instead of encrypted with the recipient RSA public

key) and the session key K , inside the ticket private part, is now encrypted using the shared key between the issuer and the ticket target entity. As for the session setup protocol, the only change is that the target entity will recover the session key K from the ticket private part using the shared key with its issuer, the A3C server.

7.3.2 GW

Similarly to the DH, the GW (implemented in a Raspberry Pi device) is also a target in the session protocol, establishing sessions with the DHMs and Clients of the system. This way it also performs the same asymmetric operations:

- One RSA signature validation of the received authorization ticket signature.
- One RSA decryption of the ticket private part, to obtain the ticket session key.

The results of comparing the 2048 bit RSA operations with the ECC algorithms, in a total of 10000, tests are present in Table 7.2. Here it is possible to observe that, in the same fashion as in the DH tests, the RSA signature validation is the fastest of the considered algorithms, while the RSA decryption is still considerably slow. However, the GWs, although computational restricted, are still faster than the DHs and the performance of the algorithms is not as critical. Still, as they are required to support dozens of parallel connections with multiple DHs, performance is still an important factor.

Algorithm	Decrypt (s)	Sign validation (s)
RSA	2.775872e-2	1.285911e-3
ECDSA SECP224R1		3.921158e-3
ECDSA SECP256R1		1.631199e-3
ECDSA SECP384R1		1.302308e-2

Table 7.2: Raspberry pi performance comparison between RSA and different ECDSA curves. Signature validation operations were performed over 256 bytes of data. Decryption operations were performed over 256 bytes of data.

7.3.3 Client, A3Cs and DHM

The Client, A3Cs and DHM were all implemented and executed on a laptop with high-end specs, and as such the performance here is not as critical as in the GW or the DH. Nonetheless the algorithms were still tested as these entities, namely the A3Cs, perform more asymmetrical key operations than the remaining entities. The results, obtained from a total of 10000 tests,

Algorithm	Encrypt (s)	Decrypt (s)	Sign (s)	Sign validation (s)
RSA	3.837521e-5	7.317466e-4	6.089797e-4	5.120390e-5
ECDSA SECP224R1			9.933335e-5	1.845952e-4
ECDSA SECP256R1			3.484903e-5	7.483517e-5
ECDSA SECP384R1			1.291431e-3	4.797327e-7

Table 7.3: Laptop performance comparison between RSA and different ECDSA curves. Signature and signature validation operations were performed over 412 and 256 bytes of data, respectively. Encryption and decryption operations were performed over 16 and 256 bytes of data, respectively.

are present in Table 7.3. Given the obtained results, it is possible to conclude that SECP384R1 is the slowest algorithm for generating signatures, while also being the fastest for validating the signatures.

Conclusions

This chapter presents a brief overview of the work developed in this dissertation as well as the academic contributions it achieved. A discussion of potential improvements that could enhance the robustness of the developed infrastructure is also here presented.

8.1 FINAL CONSIDERATIONS

Proper assurance of privacy and security in the IoT are some of the key issues affecting it, mostly due to its highly distributed and heterogeneous nature. Additionally, developing software architectures for this domain with security oriented perspective is neither a trivial task, as it must consider a set of constraints, from performance to practicality and ease of integration with current environments.

This way, this dissertation primary objective was the development and consequential prototype deployment of a proof of concept IoT architecture with security and privacy considerations.

In order to accomplish such goal, the exploration of the IoT concept was required, along with its adopted technologies, architectures, and issues, while also exploring some of the already existing solutions.

The result was a security oriented architecture with separation of duties and fine-grained access control in the interactions with the IoT devices and their data, preventing them from being manipulated by attackers, while avoiding the leakage of data to unauthorized endpoints. For this, no PKI is used and the access control is performed in a split approach with third-party entities that create and manage the policies to be enforced by the entities they oversee. The underlying authentication is performed through the issuing of access tickets, in a Kerberos-like fashion, used to establish secure sessions between the network entities.

The developed prototype aimed to implement such architecture using suitable hardware and technologies for IoT systems. This way, BLE was chosen as the underlying low-cost communication technology for connecting the IoT devices and Protocol Buffers were adopted as the mechanism for inter-domain data serialization, that is, between the GWs and DHs. As

for the communication of the IP-addressable entities, each implements its own web server and data is transferred using the classic HTTP model, through a well defined REST API.

8.2 FUTURE WORK

Although the main objectives of this dissertation were fulfilled, as with every proof-of-concept project there is still a lot of work that could be done to improve the current architecture and its respective prototype. The following list identifies some of the most relevant issues to be considered in the future development of this work:

- Adopt CoAP as the reference protocol for lightweight communication between IP-based entities and compare its performance against the adopted REST standard.
- Replacement of the used RSA algorithms with ones based on elliptic curves, evaluating and comparing their performance.
- Development of a protocol to properly handle the revocation and refreshment of secure sessions.
- Rule definition at A3Cs for entity validation during ticket issuing.
- Study and consider the deployment of some system entities outside the local network scope, mainly the Client applications, which are usually running remotely, such as in the cloud.
- "Dockerize" each network entity in order to ease their deployment and configuration.

References

- [1] 1Password, *How PBKDF2 strengthens your Master Password*, [Accessed: September 2019], Jun. 2019. [Online]. Available: <https://support.1password.com/pbkdf2/>.
- [2] Admin, *The History of IoT: a Comprehensive Timeline of Major Events, Infographic*, [Accessed: November 2018]. [Online]. Available: <https://hqsoftwarelab.com/about-us/blog/the-history-of-iot-a-comprehensive-timeline-of-major-events-infographic>.
- [3] P. Aimonen, *Nanopb - protocol buffers with small code size*, [Accessed: October 2019]. [Online]. Available: <https://jpa.kapsi.fi/nanopb/>.
- [4] I. F. Akyildiz, T. Melodia, and K. R. Chowdhury, "A survey on wireless multimedia sensor networks", *Computer networks*, vol. 51, no. 4, pp. 921–960, 2007.
- [5] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey", *Computer networks*, vol. 38, no. 4, pp. 393–422, 2002.
- [6] Amazon, *Amazon Cognito Identities - AWS IoT*, [Accessed: June 2019]. [Online]. Available: <http://docs.aws.amazon.com/iot/latest/developerguide/cognito-identities.html>.
- [7] —, *Authorization - AWS IoT*, [Accessed: June 2019]. [Online]. Available: <http://docs.aws.amazon.com/iot/latest/developerguide/authorization.html>.
- [8] —, *AWS IoT framework*, [Accessed: June 2019]. [Online]. Available: <https://aws.amazon.com/iot>.
- [9] —, *IAM, Users, Groups and Roles - AWS IoT*, [Accessed: June 2019]. [Online]. Available: <http://docs.aws.amazon.com/iot/latest/developerguide/iam-users-groups-roles.html>.
- [10] M. Ammar, G. Russello, and B. Crispo, "Internet of Things: A survey on the security of IoT frameworks", *Journal of Information Security and Applications*, vol. 38, pp. 8–27, 2018.
- [11] M. Andersson, "Use case possibilities with Bluetooth low energy in IoT applications", u-blox, White paper UBX-14054580, May 2014.
- [12] K. Ashton, *That 'Internet of Things' Thing*, [Accessed: November 2018], 2009. [Online]. Available: <https://www.rfidjournal.com/articles/view?4986>.
- [13] Y. Atwady and M. Hammoudeh, "A survey on authentication techniques for the internet of things", in *Proceedings of the International Conference on Future Networks and Distributed Systems*, ACM, 2017, p. 8.
- [14] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey", *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.
- [15] J.-P. Aumasson, S. Neves, Z. Wilcox-O’Hearn, and C. Winnerlein, "BLAKE2: simpler, smaller, fast as MD5", in *International Conference on Applied Cryptography and Network Security*, Springer, 2013, pp. 119–135.
- [16] Axiomatics, *Stack Overflow: Creating a consistent authorization framework*, <https://www.axiomatics.com/blog/stack-overflow-creating-a-consistent-authorization-framework/>, [Accessed: November 2019], Apr. 2019.

- [17] Axway, *Kerberos authentication*, [Accessed: June 2019]. [Online]. Available: https://docs.axway.com/bundle/APIGateway_762_IntegrationKerberos_allOS_en_HTML5/page/Content/KerberosIntegration/kerberos_overview.htm.
- [18] V. D. B. Russel, *Practical Internet of Things Security*. Packt Publishing Ltd., 2016, Chapter 1.
- [19] B. Stewart, *Internet History – One Page Summary*, [Accessed: November 2018], 2000. [Online]. Available: https://www.livinginternet.com/i/ii_summary.htm.
- [20] E. Barker and Q. Dang, “Recommendation for Key Management - Part 3: Application-Specific Key Management Guidance”, NIST, Technical Report 800-57, Jan. 2015.
- [21] M. Bhargava, *IoT Projects with Bluetooth Low Energy*. Packt, 30 August 2017. [Online]. Available: https://subscription.packtpub.com/book/hardware_and_creative/9781788399449/1/011v11sec11/architecture-of-bluetooth-low-energy.
- [22] D. Bhattacharyya, R. Ranjan, F. Alisherov, M. Choi, *et al.*, “Biometric authentication: A review”, *International Journal of u-and e-Service, Science and Technology*, vol. 2, no. 3, pp. 13–28, 2009.
- [23] M. Bon, *A Basic Introduction to BLE Security*, [Accessed: October 2019], Oct. 2016. [Online]. Available: <https://www.digikey.com/eewiki/display/Wireless/A+Basic+Introduction+to+BLE+Security>.
- [24] W. J. Buchanan, S. Li, and R. Asif, “Lightweight cryptography methods”, *Journal of Cyber Security Technology*, vol. 1, no. 3-4, pp. 187–201, 2017.
- [25] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, “Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility”, *Future Generation computer systems*, vol. 25, no. 6, pp. 599–616, 2009.
- [26] K.-H. Chang, “Bluetooth: a viable solution for IoT?[Industry Perspectives]”, *IEEE Wireless Communications*, vol. 21, no. 6, pp. 6–7, 2014.
- [27] L. Da Xu, W. He, and S. Li, “Internet of things in industries: A survey”, *IEEE Transactions on industrial informatics*, vol. 10, no. 4, pp. 2233–2243, 2014.
- [28] M. Darianian and M. P. Michael, “Smart home mobile RFID-based Internet-of-Things systems and services”, in *2008 International conference on advanced computer theory and engineering*, IEEE, 2008, pp. 116–120.
- [29] B. Dorsemayne, J.-P. Gaulier, J.-P. Wary, N. Kheir, and P. Urien, “Internet of Things: a definition & taxonomy”, in *2015 9th International Conference on Next Generation Mobile Applications, Services and Technologies*, IEEE, 2015, pp. 72–77.
- [30] *Eclipse Mosquitto™ - An open source MQTT broker*, <https://mosquitto.org/>, [Accessed: November 2019].
- [31] *Eclipse Paho - MQTT and MQTT-SN software*, <https://www.eclipse.org/paho/>, [Accessed: November 2019].
- [32] Elprocus, *ZigBee Wireless Technology Architecture and Applications*, [Accessed: June 2019]. [Online]. Available: <https://www.elprocus.com/what-is-zigbee-technology-architecture-and-its-applications/>.
- [33] EPCglobal, “Object Naming Service (ONS)”, EPCglobal, Tech. Rep., Oct. 2005.
- [34] C. Faulkner, *What is NFC? Everything you need to know*, [Accessed: June 2019], May 2017. [Online]. Available: <https://www.techradar.com/news/what-is-nfc>.
- [35] K. Fawaz, K.-H. Kim, and K. G. Shin, “Protecting Privacy of BLE Device Users”, in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 1205–1221.
- [36] M. Froese, *Global IoT market to reach \$318 billion by 2023, says GlobalData*, [Accessed: November 2018], Windpower Engineering, November 19, 2018. [Online]. Available: <https://www.windpowerengineering.com/global-iot-market-to-reach-318-billion-by-2023-says-globaldata/>.
- [37] *Gartner Says 8.4 Billion Connected "Things" Will Be in Use in 2017, Up 31 Percent From 2016*, <https://www.gartner.com/en/newsroom/press-releases/2017-02-07-gartner-says-8-billion>

connected-things-will-be-in-use-in-2017-up-31-percent-from-2016, [Accessed: November 2018], Feb. 2017.

- [38] D. Genkin, L. Pachmanov, I. Pipman, and E. Tromer, “Stealing keys from PCs using a radio: Cheap electromagnetic attacks on windowed exponentiation”, in *International workshop on cryptographic hardware and embedded systems*, Springer, 2015, pp. 207–228.
- [39] —, “ECDH key-extraction via low-bandwidth electromagnetic attacks on PCs”, in *Cryptographers’ Track at the RSA Conference*, Springer, 2016, pp. 219–235.
- [40] K. Gill, S.-H. Yang, F. Yao, and X. Lu, “A zigbee-based home automation system”, *IEEE Transactions on consumer Electronics*, vol. 55, no. 2, pp. 422–430, 2009.
- [41] H. Gochkov, I. Grokhtkov, and L. Bernstone, *FFat*, GitHub repository, <https://github.com/espressif/arduino-esp32/tree/master/libraries/FFat>.
- [42] Google, *Protocol Buffers*, [Accessed: October 2019]. [Online]. Available: <https://developers.google.com/protocol-buffers>.
- [43] Q. Gou, L. Yan, Y. Liu, and Y. Li, “Construction and strategies in IoT security system”, in *2013 IEEE international conference on green computing and communications and IEEE internet of things and IEEE cyber, physical and social computing*, IEEE, 2013, pp. 1129–1132.
- [44] GSM Association, “IoT Security Guidelines”, GSMA, Tech. Rep., 31 October 2017.
- [45] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, “Internet of Things (IoT): A vision, architectural elements, and future directions”, *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.
- [46] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio, “Interacting with the soa-based internet of things: Discovery, query, selection, and on-demand provisioning of web services”, *IEEE transactions on Services Computing*, vol. 3, no. 3, pp. 223–235, 2010.
- [47] M. El-hajj, A. Fadlallah, M. Chamoun, and A. Serhrouchni, “A survey of internet of things (IoT) Authentication schemes”, *Sensors*, vol. 19, no. 5, p. 1141, 2019.
- [48] C. Han, J. M. Jornet, E. Fadel, and I. F. Akyildiz, “A cross-layer communication module for the Internet of Things”, *Computer Networks*, vol. 57, no. 3, pp. 622–633, 2013.
- [49] D. Hardt, *The OAuth 2.0 Authorization Framework*, RFC 6749 (Proposed Standard), Internet Engineering Task Force, Oct. 2012. [Online]. Available: <http://www.ietf.org/rfc/rfc6749.txt>.
- [50] I. Harvey, *Bluepy*, <https://github.com/IanHarvey/bluepy>, 2013.
- [51] H. He, “What is service-oriented architecture”, *Publicação eletrônica em*, vol. 30, pp. 1–5, 2003.
- [52] HP, *HP Study Reveals 70 Percent of Internet of Things Devices Vulnerable to Attack*, [Accessed: March 2019], 29 July 2014. [Online]. Available: <https://www8.hp.com/us/en/hp-news/press-release.html?id=1744676>.
- [53] F. Hu, *Security and privacy in Internet of things (IoTs): Models, Algorithms, and Implementations*. CRC Press, 2016.
- [54] P. Hunter, *IoT + NFC: Four reasons why IoT needs NFC*, [Accessed: June 2019], Aug. 2016. [Online]. Available: <https://internetofthingsagenda.techtarget.com/blog/IoT-Agenda/IoT-NFC-Four-reasons-why-IoT-needs-NFC>.
- [55] J. Ibarra-Esquer, F. González-Navarro, B. Flores-Rios, L. Burtseva, and M. Astorga-Vargas, “Tracking the evolution of the internet of things concept across different application domains”, *Sensors*, vol. 17, no. 6, p. 1379, 2017.
- [56] E. Ilie-Zudor, Z. Kemény, F. Van Blommestein, L. Monostori, and A. Van Der Meulen, “A survey of applications and requirements of unique identification systems and RFID techniques”, *Computers in Industry*, vol. 62, no. 3, pp. 227–252, 2011.

- [57] Interagency International Cybersecurity Standardization Working Group ((IICS WG)), “Interagency Report on the Status of International Cybersecurity Standardization for the Internet of Things (IoT)”, NIST, NISTIR 8200, Nov. 2018.
- [58] International Organization for Standardization/International Electrotechnical Commission and others, “ISO/IEC 11889-1:2009 – Information technology – Trusted Platform Module – Part 1: Overview”, *International Standard, ISO/IEC*, pp. 11 889–1, May 2009.
- [59] IoT Security Foundation, “Secure Design Best Practices Guide release 1.2.1”, IoTSF, Tech. Rep., Dec. 2018.
- [60] R. H. Jhaveri, N. M. Patel, Y. Zhong, and A. K. Sangaiah, “Sensitivity analysis of an attack-pattern discovery based trusted routing scheme for mobile ad-hoc networks in industrial IoT”, *IEEE Access*, vol. 6, pp. 20 085–20 103, 2018.
- [61] X. Jia, Q. Feng, T. Fan, and Q. Lei, “RFID technology and its applications in Internet of Things (IoT)”, in *2012 2nd international conference on consumer electronics, communications and networks (CECNet)*, IEEE, 2012, pp. 1282–1285.
- [62] B. Kaliski, *Public-Key Cryptography Standards (PKCS) #8: Private-Key Information Syntax Specification Version 1.2*, RFC 5208 (Informational), Obsoleted by RFC 5958, Internet Engineering Task Force, May 2008. [Online]. Available: <http://www.ietf.org/rfc/rfc5208.txt>.
- [63] M. Katagi, S. Moriai, *et al.*, “Lightweight cryptography for the internet of things”, *Sony Corporation*, pp. 7–10, 2008.
- [64] J. Katz, A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*. CRC press, 1996.
- [65] M. A. Khan and K. Salah, “IoT security: Review, blockchain solutions, and open challenges”, *Future Generation Computer Systems*, vol. 82, pp. 395–411, 2018.
- [66] P. Kinney *et al.*, “Zigbee technology: Wireless control that simply works”, in *Communications design conference*, vol. 2, 2003, pp. 1–7.
- [67] P. Kocher, J. Jaffe, and B. Jun, “Differential power analysis”, in *Annual International Cryptology Conference*, Springer, 1999, pp. 388–397.
- [68] J. Kohl and C. Neuman, *The Kerberos Network Authentication Service (V5)*, RFC 1510 (Historic), Obsoleted by RFCs 4120, 6649, Internet Engineering Task Force, Sep. 1993. [Online]. Available: <http://www.ietf.org/rfc/rfc1510.txt>.
- [69] N. Kolban, *ESP32 BLE Arduino*, <https://github.com/IanHarvey/bluepy>, 2017.
- [70] E. A. Kosmatos, N. D. Tselikas, and A. C. Boucouvalas, “Integrating RFIDs and smart objects into a UnifiedInternet of Things architecture”, *Advances in Internet of Things*, vol. 1, no. 01, p. 5, 2011.
- [71] N. Kumar, *Enterprise Benefits on Service Oriented Architecture – SOA*, [Accessed: March 2019], 28 March 2013. [Online]. Available: <https://www.javacodegeeks.com/2013/03/enterprise-benefits-on-service-oriented-architecture-soa.html>.
- [72] P. Kumar, S. Ranganath, H. Weimin, and K. Sengupta, “Framework for real-time behavior interpretation from traffic video”, *IEEE Transactions on Intelligent Transportation Systems*, vol. 6, no. 1, pp. 43–53, 2005.
- [73] S. Kumar, *Evolution of Internet of Things(IoT)*, [Accessed: December 2018], Oct. 2017. [Online]. Available: <https://codeforbillion.blogspot.com/2017/10/evolution-of-internet-of-thingsiot.html>.
- [74] N. Kushalnagar, G. Montenegro, and C. Schumacher, *IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals*, RFC 4919 (Informational), Internet Engineering Task Force, Aug. 2007. [Online]. Available: <http://www.ietf.org/rfc/rfc4919.txt>.
- [75] J.-S. Lee, Y.-W. Su, C.-C. Shen, *et al.*, “A comparative study of wireless protocols: Bluetooth, UWB, ZigBee, and Wi-Fi”, *Industrial electronics society*, vol. 5, pp. 46–51, 2007.

- [76] B. Li and J. Yu, "Research and application on the smart home based on component technologies and Internet of Things", *Procedia Engineering*, vol. 15, pp. 2087–2092, 2011.
- [77] H.-E. Lin, R. Zito, M. Taylor, *et al.*, "A review of travel-time prediction in transport and logistics", in *Proceedings of the Eastern Asia Society for transportation studies*, vol. 5, 2005, pp. 1433–1448.
- [78] Z. Liqiang, Y. Shouyi, L. Leibo, Z. Zhen, and W. Shaojun, "A crop monitoring system based on wireless sensor network", *Procedia Environmental Sciences*, vol. 11, pp. 558–565, 2011.
- [79] C. H. Liu, B. Yang, and T. Liu, "Efficient naming, addressing and profile services in Internet-of-Things sensory environments", *Ad Hoc Networks*, vol. 18, pp. 85–101, 2014.
- [80] R. Loladia, *Elliptic Curve Cryptography and Forward Secrecy Support in AWS IoT*, [Accessed: June 2019], Amazon, 20 May 2016. [Online]. Available: <https://aws.amazon.com/blogs/iot/elliptic-curve-cryptography-and-forward-secrecy-support-in-aws-iot-3/>.
- [81] H. Luo, S. Ci, D. Wu, N. Stergiou, and K.-C. Siu, "A remote markerless human gait tracking for e-healthcare based on content-aware wireless multimedia communications", *IEEE Wireless Communications*, vol. 17, no. 1, pp. 44–50, 2010.
- [82] M. Maheshwari, *Understanding SSH workflow*, [Accessed: September 2019], Sep. 2017. [Online]. Available: https://medium.com/@Magical_Mudit/understanding-ssh-workflow-66a0e8d4bf65.
- [83] C. P. Mayer, "Security and privacy challenges in the internet of things", *Electronic Communications of the EASST*, vol. 17, 2009.
- [84] K. A. McKay, L. Bassham, M. S. Turan, and N. Mouha, "NISTIR 8114 report on lightweight cryptography", *National Institute of Standards and Technology (NIST)*, Gaithersburg, 2017.
- [85] P. Mell, T. Grance, *et al.*, "The NIST definition of cloud computing", 2011.
- [86] Microsoft, *Azure IoT - The Internet of Things (IoT) for every business*, [Accessed: June 2019]. [Online]. Available: <https://azure.microsoft.com/en-us/overview/iot/>.
- [87] —, *Power BI, Interactive Data Visualization BI Tools*, [Accessed: June 2019]. [Online]. Available: <https://powerbi.microsoft.com>.
- [88] R. Minerva, A. Biru, and D. Rotondi, "Towards a definition of the Internet of Things (IoT)", *IEEE Internet Initiative*, vol. 1, pp. 1–86, 2015.
- [89] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, "Internet of things: Vision, applications and research challenges", *Ad hoc networks*, vol. 10, no. 7, pp. 1497–1516, 2012.
- [90] D. Moeinfar, H. Shamsi, and F. Nafar, "Design and implementation of a low-power active RFID for container tracking at 2.4 GHz frequency", *Advances in Internet of Things*, vol. 2, no. 02, p. 13, 2012.
- [91] MQTT.org, *Frequently Asked Questions*, [Accessed: June 2019]. [Online]. Available: <http://mqtt.org/faq>.
- [92] M. F. Muhammad, W. Anjum, and K. S. Mazhar, "A Critical Analysis on the Security Concerns of Internet of Things (IoT)", *International Journal of Computer Applications (0975 8887)*, vol. 111, no. 7, 2015.
- [93] N. Naik, "Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP", in *2017 IEEE international systems engineering symposium (ISSE)*, IEEE, 2017, pp. 1–7.
- [94] R. M. Needham and M. D. Schroeder, "Using encryption for authentication in large networks of computers", *Communications of the ACM*, vol. 21, no. 12, pp. 993–999, 1978.
- [95] P. Nohe, *The Difference Between SHA-1, SHA-2 and SHA-256 Hash Algorithm*, [Accessed: June 2019], Sep. 2018. [Online]. Available: <https://www.thesslstore.com/blog/difference-sha-1-sha-2-sha-256-hash-algorithms/>.
- [96] M. O'Neill *et al.*, "Insecurity by design: Today's IoT device security problem", *Engineering*, vol. 2, no. 1, pp. 48–49, 2016.

- [97] OASIS, *OASIS eXtensible Access Control Markup Language (XACML) TC*, [Accessed: June 2019]. [Online]. Available: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml.
- [98] J. Olenski, *ECC 101: What is ECC and why would I want to use it?*, <https://www.globalsign.com/en/blog/elliptic-curve-cryptography/>, [Accessed: October 2019], May 2015.
- [99] oneM2M, “Security”, Technical Report TR-0008-V 2.0.1, 27 February 2018.
- [100] *What is OpenID?*, <https://openid.net/what-is-openid/>, [Accessed: June 2019].
- [101] OWASP, *OWASP Internet of Things Project*, [Accessed: May 2019]. [Online]. Available: https://www.owasp.org/index.php/OWASP_Internet_of_Things_Project#tab=Main.
- [102] S. Pfleeger and R. Cunningham, “Why measuring security is hard”, *IEEE Security & Privacy*, vol. 8, no. 4, pp. 46–54, 2010.
- [103] M. Pinola, *Bluetooth Basics - What Bluetooth is, what it does, and how it works*, [Accessed: June 2019], Feb. 2019. [Online]. Available: <https://www.lifewire.com/what-is-bluetooth-2377412>.
- [104] M. Presser, *The rise of IoT - why today?*, [Accessed: November 2018], January 12, 2016. [Online]. Available: <https://iot.ieee.org/newsletter/january-2016/the-rise-of-iot-why-today.html>.
- [105] Promow, *Amazon Web Services – IoT*, [Accessed: June 2019]. [Online]. Available: <http://www.promow.be/resources/amazon-web-services-iot/>.
- [106] Published by CCM user "aakai056", *What is WiFi and How Does it Work?*, [Accessed: June 2019]. [Online]. Available: <https://ccm.net/faq/298-what-is-wifi-and-how-does-it-work>.
- [107] A. Purcell, *IBM - 3 key ideas to help drive compliance in the cloud*, [Accessed: May 2019], 16 January 2018. [Online]. Available: <https://www.ibm.com/blogs/cloud-computing/2018/01/16/drive-compliance-cloud/>.
- [108] Recommendation ITU-T Y.4455, *Reference architecture for Internet of things network service capability exposure*, Oct. 2017. [Online]. Available: <https://www.itu.int/rec/T-REC-Y.4455-201710-I/en>.
- [109] E. Rescorla and N. Modadugu, *Datagram Transport Layer Security Version 1.2*, RFC 6347 (Proposed Standard), Internet Engineering Task Force, Jan. 2012. [Online]. Available: <http://www.ietf.org/rfc/rfc6347.txt>.
- [110] D. R. Roberto Minerva Abyi Biru, “Towards a definition of the Internet of Things (IoT)”, IEEE, Tech. Rep., May 2015.
- [111] P. Romano, *A Bluetooth Low Energy Application: Defining the BLE Stack*, [Accessed: March 2019], Dec. 2014. [Online]. Available: <https://www.semiconductorstore.com/blog/2014/A-Bluetooth-Low-Energy-Application-Defining-the-BLE-Stack/883/>.
- [112] T. Rosa, “Bypassing Passkey Authentication in Bluetooth Low Energy.”, *IACR Cryptology ePrint Archive*, vol. 2013, p. 309, 2013.
- [113] M. Rouse, *What is confidentiality, integrity, and availability (CIA triad)?*, [Accessed: March 2019], Nov. 2014. [Online]. Available: <https://whatis.techtarget.com/definition/Confidentiality-integrity-and-availability-CIA>.
- [114] M. Saadeh, A. Sleit, M. Qatawneh, and W. Almobaideen, “Authentication techniques for the internet of things: A survey”, in *2016 Cybersecurity and Cyberforensics Conference (CCC)*, IEEE, 2016, pp. 28–34.
- [115] P. Saint-Andre and J. Klensin, “Uniform Resource Names (URNs)”, RFC Editor, RFC 8141, Apr. 2017.
- [116] K. Sakamura, “Challenges in the age of ubiquitous computing: a case study of T-Engine, an open development platform for embedded systems”, in *Proceedings of the 28th international conference on Software engineering*, ACM, 2006, pp. 713–720.
- [117] F. Samie, L. Bauer, and J. Henkel, “IoT technologies for embedded computing: A survey”, in *Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, ACM, 2016, p. 8.

- [118] R. Shahan and B. Lamos, *Security for Internet of Things (IoT) from the ground up*, [Accessed: June 2019], Microsoft, Oct. 2018. [Online]. Available: <https://docs.microsoft.com/en-us/azure/iot-fundamentals/iot-security-ground-up>.
- [119] Shancang Li, Li Da Xu, *Securing the Internet of Things*, 1st Edition. Syngress, Jan. 2017, Chapter 4 - IoT node authentication.
- [120] Z. Shelby, K. Hartke, and C. Bormann, *The Constrained Application Protocol (CoAP)*, RFC 7252 (Proposed Standard), Internet Engineering Task Force, Jun. 2014. [Online]. Available: <http://www.ietf.org/rfc/rfc7252.txt>.
- [121] SIG, "Bluetooth core specification version 4.2", URL <https://www.bluetooth.com/specifications/bluetooth-core-specification>, Dec. 2014.
- [122] K. Sovani, *Understanding ESP32's Security Features*, [Accessed: October 2019], Jun. 2018. [Online]. Available: <https://medium.com/the-esp-journal/understanding-esp32s-security-features-14483e465724>.
- [123] *SSL Library mbed TLS / Polar SSL*, <https://tls.mbed.org>, [Accessed: October 2019].
- [124] SSL2buy, *Symmetric vs. Asymmetric Encryption – What are differences?*, [Accessed: July 2019]. [Online]. Available: <https://www.ssl2buy.com/wiki/symmetric-vs-asymmetric-encryption-what-are-differences>.
- [125] M. Stanislav and T. Beardsley, "Hacking iot: A case study on baby monitor exposures and vulnerabilities", *Rapid* 7, 2015.
- [126] B. Stewart, *IPTO – Information Processing Techniques Office*, [Accessed: November 2018], 2000. [Online]. Available: https://www.livinginternet.com/i/ii_ipto.htm.
- [127] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation", in *2007 44th ACM/IEEE Design Automation Conference*, IEEE, 2007, pp. 9–14.
- [128] C. Sun, "Application of RFID technology for logistics on internet of things", *AASRI Procedia*, vol. 1, pp. 106–111, 2012.
- [129] B. Sunar, W. J. Martin, and D. R. Stinson, "A provably secure true random number generator with built-in tolerance to active attacks", *IEEE Transactions on computers*, vol. 56, no. 1, pp. 109–119, 2006.
- [130] H. Suo, J. Wan, C. Zou, and J. Liu, "Security in the internet of things: A review", in *2012 international conference on computer science and electronics engineering*, IEEE, vol. 3, 2012, pp. 648–651.
- [131] Symantec Corporation, *2018 Internet Security Threat Report*, [Accessed: March 2019], Mar. 2018. [Online]. Available: <https://www.symantec.com/content/dam/symantec/docs/reports/istr-23-executive-summary-en.pdf>.
- [132] *The little-known story of the first IoT device*, [Accessed: November 2018], Feb. 2018. [Online]. Available: <https://www.ibm.com/blogs/industries/little-known-story-first-iot-device/>.
- [133] Tornado, *Tornado Web Server*, [Accessed: September 2019]. [Online]. Available: <https://www.tornadoweb.org/en/stable/>.
- [134] S. Tozlu, "Feasibility of Wi-Fi enabled sensors for Internet of Things", in *2011 7th International Wireless Communications and Mobile Computing Conference*, IEEE, 2011, pp. 291–296.
- [135] S. Tozlu, M. Senel, W. Mao, and A. Keshavarzian, "Wi-Fi enabled sensors for internet of things: A practical approach", *IEEE Communications Magazine*, vol. 50, no. 6, pp. 134–143, 2012.
- [136] S. Turner and L. Chen, *Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms*, RFC 6151 (Informational), Internet Engineering Task Force, Mar. 2011. [Online]. Available: <http://www.ietf.org/rfc/rfc6151.txt>.
- [137] Tutorialspoint, *Software Architecture & Design Introduction*, [Accessed: March 2019]. [Online]. Available: https://www.tutorialspoint.com/software_architecture_design/introduction.htm.

- [138] M. Weiser, R. Gold, and J. S. Brown, “The origins of ubiquitous computing research at PARC in the late 1980s”, *IBM systems journal*, vol. 38, no. 4, pp. 693–696, 1999.
- [139] *Welcome to pyca/cryptography*, <https://cryptography.io>, [Accessed: October 2019].
- [140] *What is RFID and How Does RFID Work?*, [Accessed: November 2018], American Barcode and RFID (AB&R). [Online]. Available: <https://www.abr.com/what-is-rfid-how-does-rfid-work/>.
- [141] A. Whitmore, A. Agarwal, and L. Da Xu, “The Internet of Things—A survey of topics and trends”, *Information Systems Frontiers*, vol. 17, no. 2, pp. 261–274, 2015.
- [142] Wind River Systems, *Security in Internet of Things*, 2015. [Online]. Available: https://www.windriver.com/whitepapers/security-in-the-internet-of-things/wr_security-in-the-internet-of-things.pdf.
- [143] wolfSSL, *What is a Block Cipher?*, [Accessed: July 2019], 19 December 2014. [Online]. Available: <https://www.wolfssl.com/what-is-a-block-cipher/>.
- [144] M. Wu, T.-J. Lu, F.-Y. Ling, J. Sun, and H. Du, “Research on the architecture of Internet of Things”, vol. 5, Sep. 2010, pp. V5–484. DOI: 10.1109/ICACTE.2010.5579493.
- [145] Y. Wu, Q. Z. Sheng, and S. Zeadally, “RFID: opportunities and challenges”, in *Next-generation wireless technologies*, Springer, 2013, pp. 105–129.
- [146] J. Wurm, K. Hoang, O. Arias, A.-R. Sadeghi, and Y. Jin, “Security analysis on consumer and industrial IoT devices”, in *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, IEEE, 2016, pp. 519–524.
- [147] M. Yun and B. Yuxin, “Research on the architecture and key technology of Internet of Things (IoT) applied on smart grid”, in *2010 International Conference on Advances in Energy Engineering*, IEEE, 2010, pp. 69–72.
- [148] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, P. Crowley, C. Papadopoulos, L. Wang, B. Zhang, *et al.*, “Named data networking”, *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 66–73, 2014.
- [149] Z.-K. Zhang, M. C. Y. Cho, and S. Shieh, “Emerging security threats and countermeasures in IoT”, in *Proceedings of the 10th ACM symposium on information, computer and communications security*, ACM, 2015, pp. 1–6.
- [150] Z.-K. Zhang, M. C. Y. Cho, C.-W. Wang, C.-W. Hsu, C.-K. Chen, and S. Shieh, “IoT security: ongoing challenges and research opportunities”, in *2014 IEEE 7th international conference on service-oriented computing and applications*, IEEE, 2014, pp. 230–234.
- [151] K. Zhao and L. Ge, “A survey on the internet of things security”, in *2013 Ninth international conference on computational intelligence and security*, IEEE, 2013, pp. 663–667.
- [152] M. Zorzi, A. Gluhak, S. Lange, and A. Bassi, “From today’s intranet of things to a future internet of things: a wireless-and mobility-related view”, *IEEE Wireless communications*, vol. 17, no. 6, pp. 44–51, 2010.
- [153] A. Zúquete, H. Gomes, J. Amaral, and C. Oliveira, “Security-Oriented Architecture for Managing IoT Deployments”, *Symmetry*, vol. 11, no. 10, p. 1315, 2019.

Protocol buffer configuration file

```
1  syntax="proto2";
2
3  message BLEmessage{
4      required Header header = 1; // indicator of the message type
5      required Payload payload = 2; // contents of the message
6  }
7
8  message Header{
9      enum MessageType{
10         AuthTicket = 1;
11         NonceExchange = 2;
12         DriversInfo = 3;
13         DeviceAccessRequest = 4;
14         DeviceAccessResponse = 5;
15         ErrorMsg = 6;
16         DriverAuthTicket=7;
17     }
18
19     required MessageType msg_type = 1;
20 }
21
22 message Payload{
23     optional AuthenticationTicket auth_ticket = 1;
24     optional NonceExchangeMessage nonce_exc_msg = 2;
25     optional DriversInfoMessage drivers_info_msg = 3;
26     optional MessageResponse response = 4;
27     optional MessageRequest request = 5;
28     optional ErrorMessage error_msg = 6;
29 }
30
31 message MessageRequest{
32     required string target_uuid = 1;
33     required string action = 2;
34     optional int32 delta = 3;
```

```

35  }
36
37  message MessageResponse{
38      required string target_uuid = 1;
39      required string response = 2;
40  }
41
42  message AuthenticationTicket {
43      required bytes secret = 1;
44      required bytes pub = 2;
45      required bytes signature = 3;
46      optional bytes auth_nonce = 4;
47  }
48
49  message NonceExchangeMessage{
50      required bytes enc_nonce = 1;
51      optional bytes clear_nonce = 2;
52  }
53
54  message ErrorMessage{
55      required bytes error_code = 1;
56      optional string error_description = 2;
57  }
58
59  message DriversInfoMessage{
60      repeated DriverInfo driver_info = 1;
61  }
62
63  message DriverInfo{
64      required string api_description = 1;
65      required bytes public_key = 2;
66      required bytes a3c_dd_uuid = 3;
67  }

```

Code 1: BLEmessage.proto file specification

A3C GW debug log

```

1 2019-11-29 18:04:17,308 [MainThread ] [INFO ] A3C GW server starting
2 2019-11-29 18:04:17,308 [MainThread ] [INFO ] Loading database
3 2019-11-29 18:04:17,309 [MainThread ] [INFO ] Fetching hashed password from database
4 2019-11-29 18:04:17,492 [MainThread ] [INFO ] Correct bootstrap password
5 2019-11-29 18:04:17,493 [MainThread ] [INFO ] Public key loaded successfully
6 2019-11-29 18:04:17,494 [MainThread ] [INFO ] Private key loaded successfully
7 2019-11-29 18:04:17,494 [MainThread ] [INFO ] A3C GW Server UUID:
  ↪ c81306517f41a51b92445538e3406296
8 2019-11-29 18:04:17,495 [MainThread ] [INFO ] Fetching GW info from local database
9 2019-11-29 18:04:17,495 [MainThread ] [INFO ] Successfully loaded GWs info from local
  ↪ database
10 2019-11-29 18:04:17,498 [MainThread ] [INFO ] Starting web server
11 2019-11-29 18:04:33,346 [MainThread ] [INFO ] Ticket request received
12 2019-11-29 18:04:33,346 [MainThread ] [INFO ] Entity is eligible to access target GW:
  ↪ 50433dbaa1de8cf4c381302970dd7f7f
13 2019-11-29 18:04:33,349 [MainThread ] [INFO ] Ticket generated successfully
14 2019-11-29 18:04:33,349 [MainThread ] [INFO ] Response send to entity with UUID
  ↪ d46f1dec96fb35c8407dc401109249c8
15 2019-11-29 18:04:33,349 [MainThread ] [INFO ] 200 POST /ticketFetch/ (127.0.0.1) 4.24ms
16 2019-11-29 18:04:42,861 [MainThread ] [INFO ] Ticket request received
17 2019-11-29 18:04:42,861 [MainThread ] [INFO ] Entity is eligible to access target GW:
  ↪ 50433dbaa1de8cf4c381302970dd7f7f
18 2019-11-29 18:04:42,864 [MainThread ] [INFO ] Ticket generated successfully
19 2019-11-29 18:04:42,864 [MainThread ] [INFO ] Response send to entity with UUID
  ↪ 7d7e3a36a28225c3376945fc9244f4ad
20 2019-11-29 18:04:42,865 [MainThread ] [INFO ] 200 POST /ticketFetch/ (127.0.0.1) 4.17ms

```


A3C DD debug log

```
1 2019-11-29 18:04:19,522 [MainThread ] [INFO ] A3C DD server starting
2 2019-11-29 18:04:19,523 [MainThread ] [INFO ] Loading database
3 2019-11-29 18:04:19,525 [MainThread ] [INFO ] Fetching hashed password from database
4 2019-11-29 18:04:19,719 [MainThread ] [INFO ] Correct bootstrap password
5 2019-11-29 18:04:19,719 [MainThread ] [INFO ] Public key loaded successfully
6 2019-11-29 18:04:19,721 [MainThread ] [INFO ] Private key loaded successfully
7 2019-11-29 18:04:19,722 [MainThread ] [INFO ] A3C DD Server UUID:
   ↪ ee3aeedd1bc30c4d2472d638064151c4
8 2019-11-29 18:04:19,723 [MainThread ] [INFO ] Successfully loaded GWs info from local
   ↪ database
9 2019-11-29 18:04:19,726 [MainThread ] [INFO ] Starting web server
10 2019-11-29 18:04:42,969 [MainThread ] [INFO ] Ticket request received from client uuid:
   ↪ 7d7e3a36a28225c3376945fc9244f4ad
11 2019-11-29 18:04:42,970 [MainThread ] [INFO ] Entity is eligible to access target DD:
   ↪ f17a1fe9d42b711c463cdad54e5db6f0
12 2019-11-29 18:04:42,982 [MainThread ] [INFO ] Ticket generated successfully
13 2019-11-29 18:04:42,985 [MainThread ] [INFO ] 200 POST /ticketFetch/ (127.0.0.1) 19.63ms
```


APPENDIX D

DHM debug log

```
1 2019-11-29 18:04:33,159 [MainThread ] [INFO ] DHM server starting
2 2019-11-29 18:04:33,159 [MainThread ] [INFO ] Loading database
3 2019-11-29 18:04:33,160 [MainThread ] [INFO ] Fetching hashed password from database
4 2019-11-29 18:04:33,333 [MainThread ] [INFO ] Correct bootstrap password
5 2019-11-29 18:04:33,333 [MainThread ] [INFO ] Public key loaded successfully
6 2019-11-29 18:04:33,334 [MainThread ] [INFO ] Private key loaded successfully
7 2019-11-29 18:04:33,335 [MainThread ] [INFO ] Server ID : d46f1dec96fb35c8407dc401109249c8
8 2019-11-29 18:04:33,336 [MainThread ] [INFO ] Successfully loaded DHs info from local
  ↳ database
9 2019-11-29 18:04:33,336 [MainThread ] [INFO ] Successfully loaded GWs info from local
  ↳ database
10 2019-11-29 18:04:33,337 [MainThread ] [INFO ] Starting session with GW A3C server:
  ↳ c81306517f41a51b92445538e3406296
11 2019-11-29 18:04:33,337 [Thread-1 ] [INFO ] Fetching ticket from A3C GW with uuid
  ↳ c81306517f41a51b92445538e3406296
12 2019-11-29 18:04:33,344 [MainThread ] [INFO ] Starting web server
13 2019-11-29 18:04:33,351 [Thread-1 ] [INFO ] Successfully fetched ticket from
  ↳ c81306517f41a51b92445538e3406296
14 2019-11-29 18:04:33,352 [Thread-2 ] [INFO ] Starting DHM <-> GW session. GW UUID
  ↳ 50433dbaa1de8cf4c381302970dd7f7f
15 2019-11-29 18:04:33,451 [Thread-2 ] [INFO ] Successfully established session with GW
  ↳ UUID: 50433dbaa1de8cf4c381302970dd7f7f
16 2019-11-29 18:04:33,451 [Thread-2 ] [INFO ] Fetching DHs data to configure target GW UUID
  ↳ 50433dbaa1de8cf4c381302970dd7f7f
17 2019-11-29 18:04:33,459 [Thread-2 ] [INFO ] Sending configuration tickets to target GW
  ↳ UUID 50433dbaa1de8cf4c381302970dd7f7f
18 2019-11-29 18:04:38,560 [Thread-2 ] [INFO ]
19 2019-11-29 18:04:41,986 [MainThread ] [INFO ] DH 4b97c0a6358f5da943afdc747a7c86c
  ↳ successfully configured
20 2019-11-29 18:04:41,988 [MainThread ] [INFO ] 200 POST /dhSessionConfirm/ (192.168.1.79)
  ↳ 2.08ms
```


DH debug log

```
1  rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
2  configsip: 0, SPIWP:0xee
3  clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
4  mode:DIO, clock div:1
5  load:0x3fff0018,len:4
6  load:0x3fff001c,len:928
7  ho 0 tail 12 room 4
8  load:0x40078000,len:8424
9  ho 0 tail 12 room 4
10 load:0x40080400,len:5868
11 entry 0x4008069c
12
13 File system mounted successfully
14 Total space:    1978368
15 Free space:    1966080
16 Reading keys from device local partition
17 All keys loaded successfully!
18 Starting BLE server...
19 Device host online waiting for authentication
20
21 New BLE connection received!
22 Authentication ticket received
23 Decoding private ticket section
24 Decoding public ticket section
25 Decoding signature
26 Decoding authentication nonce
27 [OK] Message decoding
28 [OK] Public RSA key parse!
29 [OK] Message digest!
30 [OK] Signature valid!
31 [OK] Authentication token is valid!
32 [OK] Private RSA key parse!
33 [OK] RSA Decryption!
34 [OK] Derived session key was digested successfully
```

35 [OK] AES CBC encryption successful
36 [OK] Message encoding
37
38 Nonce exchange response message received
39 Decoding encrypted nonce
40 [OK] Message decoding
41 [OK] AES CBC decryption successful
42 [OK] Decrypted nonce matches previously sent nonce!
43 [OK] Session established successfully with target GW
44 [OK] Message encoding
45 Message exceeds maximum link size (514). Fragmenting data...
46 Message data split into 3 fragments
47 Notifying message fragment 0...
48 Notifying message fragment 1...
49 Notifying message fragment 2...
50
51 Driver authentication ticket received!
52 Decoding private ticket section
53 Decoding public ticket section
54 Decoding signature
55 Decoding authentication nonce
56 [OK] Message decoding
57 [OK] Client target driver is temperature driver, uuid f17a1fe9d42b711c463cdad54e5db6f0
58 [OK] Public RSA key parse!
59 [OK] Message digest!
60 [OK] Signature valid!
61 [OK] Authentication token is valid!
62 [OK] Private RSA key parse!
63 [OK] RSA Decryption!
64 [OK] Secret key decryption
65 [OK] Derived session key was digested successfully
66 [OK] AES CBC encryption successful
67 [OK] Message encoding
68 Decoding target uuid field
69 Decoding driver action field
70 [OK] Message decoding
71 [OK] Client target driver is temperature driver, uuid f17a1fe9d42b711c463cdad54e5db6f0
72 Received request has delta value specification: 1
73 Temperature value (F): 123
74
75 Disconnected!

GW debug log

```

1 2019-11-29 23:13:13,799 [MainThread ] [INFO ] GW starting
2 2019-11-29 23:13:13,800 [MainThread ] [INFO ] Loading local keys
3 2019-11-29 23:13:13,801 [MainThread ] [INFO ] Public key loaded successfully
4 2019-11-29 23:13:14,245 [MainThread ] [INFO ] Private key loaded successfully
5 2019-11-29 23:13:14,246 [MainThread ] [INFO ] GW A3C public key loaded successfully
6 2019-11-29 23:13:14,248 [MainThread ] [INFO ] GW MQTT Client starting...
7 2019-11-29 23:13:14,250 [MainThread ] [INFO ] MQTT log:Sending CONNECT (u0, p0, wr0, wq0,
  ↳ wf0, c1, k60) client_id=b''
8 2019-11-29 23:13:14,252 [MainThread ] [INFO ] GW Web server starting
9 2019-11-29 23:13:14,257 [Thread-1 ] [INFO ] MQTT log:Received CONNACK (0, 0)
10 2019-11-29 23:13:14,260 [Thread-1 ] [INFO ] MQTT: New connection established
11 2019-11-29 23:13:16,030 [MainThread ] [INFO ] Session request received from DHM server
12 2019-11-29 23:13:16,033 [MainThread ] [INFO ] Received GW A3C public key is valid
13 2019-11-29 23:13:16,039 [MainThread ] [INFO ] Session ticket signature is valid
14 2019-11-29 23:13:16,081 [MainThread ] [INFO ] 200 POST /dhmSessionSetup/ (192.168.1.74)
  ↳ 70.00ms
15 2019-11-29 23:13:16,090 [MainThread ] [INFO ] Session validation request received from DHM
  ↳ server
16 2019-11-29 23:13:16,091 [MainThread ] [INFO ] Session established with DHM successfully
17 2019-11-29 23:13:16,093 [MainThread ] [INFO ] 200 POST /dhmSessionSetup/validation/
  ↳ (192.168.1.74) 4.79ms
18 2019-11-29 23:13:16,114 [MainThread ] [INFO ] Authentication tickets received from DHM
  ↳ server
19 2019-11-29 23:13:16,116 [MainThread ] [INFO ] Received message HMAC is valid
20 2019-11-29 23:13:16,117 [MainThread ] [INFO ] Scanning for nearby BLE compatible devices...
21 Discovered device: 30:ae:a4:ea:c2:c2
22 2019-11-29 23:13:21,142 [MainThread ] [INFO ] Device 30:ae:a4:ea:c2:c2 (public), RSSI=-58
  ↳ dB
23 2019-11-29 23:13:21,144 [MainThread ] [INFO ] Flags = 06
24 2019-11-29 23:13:21,145 [MainThread ] [INFO ] 0x12 = 12004000
25 2019-11-29 23:13:21,146 [MainThread ] [INFO ] Complete Local Name = DEVICE_HOST_1
26 2019-11-29 23:13:21,147 [MainThread ] [INFO ] Tx Power = 03
27 2019-11-29 23:13:21,148 [MainThread ] [INFO ] Complete 128b Services =
  ↳ 832fd58e-50c8-465b-ad1d-e6d6fccefe15

```

```

28 2019-11-29 23:13:21,152 [Thread-2 ] [INFO ] Starting session with DH 30:ae:a4:ea:c2:c2
29 2019-11-29 23:13:21,157 [MainThread ] [INFO ] 200 POST /dhTickets/ (192.168.1.74) 5045.75ms
30 2019-11-29 23:13:21,671 [Thread-2 ] [INFO ] Successfully connected to DH
    ↪ 30:ae:a4:ea:c2:c2
31 2019-11-29 23:13:22,304 [Thread-2 ] [INFO ] DH 30:ae:a4:ea:c2:c2 ready to receive
    ↪ authentication ticket
32 2019-11-29 23:13:22,390 [Thread-2 ] [INFO ] Authentication ticket message exceeds link
    ↪ maximum capacity (514) Data will be fragmented
33 2019-11-29 23:13:23,393 [Thread-2 ] [INFO ] Authentication ticket message sent to target
    ↪ DH 30:ae:a4:ea:c2:c2 successfully
34 2019-11-29 23:13:23,394 [Thread-2 ] [INFO ] Waiting for DH 30:ae:a4:ea:c2:c2 response
    ↪ notification
35 2019-11-29 23:13:23,962 [Thread-2 ] [INFO ] Notification received!
36 2019-11-29 23:13:23,964 [Thread-2 ] [INFO ] Single notification received
37 2019-11-29 23:13:23,966 [Thread-2 ] [INFO ] Nonce exchange message received
38 2019-11-29 23:13:23,969 [Thread-2 ] [INFO ] Received decrypted nonce matches session
    ↪ nonce
39 2019-11-29 23:13:23,973 [Thread-2 ] [INFO ] Session with DH 30:ae:a4:ea:c2:c2 established
    ↪ successfully
40 2019-11-29 23:13:23,975 [Thread-2 ] [INFO ] Waiting for DH 30:ae:a4:ea:c2:c2 installed
    ↪ drivers information...
41 2019-11-29 23:13:24,407 [Thread-2 ] [INFO ] Notification received!
42 2019-11-29 23:13:24,408 [Thread-2 ] [INFO ] Processing stream notification message n0
43 2019-11-29 23:13:24,412 [Thread-2 ] [INFO ] Notification received!
44 2019-11-29 23:13:24,413 [Thread-2 ] [INFO ] Processing stream notification message n1
45 2019-11-29 23:13:24,414 [Thread-2 ] [INFO ] Notification received!
46 2019-11-29 23:13:24,415 [Thread-2 ] [INFO ] Processing last stream notification
    ↪ message(n2)
47 2019-11-29 23:13:24,418 [Thread-2 ] [INFO ] Received DDs info from DH 30:ae:a4:ea:c2:c2
48 2019-11-29 23:13:24,513 [Thread-2 ] [INFO ] Waiting for further instructions...
49 2019-11-29 23:13:25,392 [MainThread ] [INFO ] Session request received from Client
50 2019-11-29 23:13:25,395 [MainThread ] [INFO ] Public key of GW A3C is valid
51 2019-11-29 23:13:25,400 [MainThread ] [INFO ] Session ticket signature is valid
52 2019-11-29 23:13:25,462 [MainThread ] [INFO ] 200 POST /clientSessionSetup/ (192.168.1.74)
    ↪ 78.59ms
53 2019-11-29 23:13:25,474 [MainThread ] [INFO ] Session validation message received from
    ↪ client
54 2019-11-29 23:13:25,475 [MainThread ] [INFO ] Session validated and established
    ↪ successfully with client
55 2019-11-29 23:13:25,476 [MainThread ] [INFO ] Sending known DDs information to client...
56 2019-11-29 23:13:25,478 [MainThread ] [INFO ] 200 POST /clientSessionSetup/validation/
    ↪ (192.168.1.74) 5.53ms
57 2019-11-29 23:13:25,512 [MainThread ] [INFO ] Session request received from client
58 2019-11-29 23:13:25,514 [MainThread ] [INFO ] Received message HMAC is valid
59 2019-11-29 23:13:25,515 [Thread-2 ] [INFO ] Client session request received
60 2019-11-29 23:13:25,517 [Thread-2 ] [INFO ] Driver authentication ticket message exceeds
    ↪ link maximum capacity (514). Data will be fragmented...
61 2019-11-29 23:13:26,520 [Thread-2 ] [INFO ] Client authentication ticket sent to target
    ↪ DH 30:ae:a4:ea:c2:c2 successfully
62 2019-11-29 23:13:26,521 [Thread-2 ] [INFO ] Waiting for DH notification...

```

```

63 2019-11-29 23:13:27,082 [Thread-2 ] [INFO ] Notification reveived!
64 2019-11-29 23:13:27,084 [Thread-2 ] [INFO ] Single notification received
65 2019-11-29 23:13:27,086 [Thread-2 ] [INFO ] Nonce exchange message received
66 2019-11-29 23:13:27,087 [Thread-2 ] [INFO ] Waiting for further instructions...
67 2019-11-29 23:13:27,090 [MainThread ] [INFO ] 200 POST /clientDDSessionSetup/
    ↳ (192.168.1.74) 1580.73ms
68 2019-11-29 23:13:27,176 [MainThread ] [INFO ] 200 POST /clientDDSessionSetup/validation/
    ↳ (192.168.1.74) 6.62ms
69 2019-11-29 23:13:27,198 [MainThread ] [INFO ] Received new client request
70 2019-11-29 23:13:27,201 [Thread-2 ] [INFO ] Subscribe client request received
71 2019-11-29 23:13:27,203 [MainThread ] [INFO ] 200 POST /clientRequest/ (192.168.1.74)
    ↳ 8.52ms
72 2019-11-29 23:13:27,207 [Thread-2 ] [INFO ] Waiting for DH response...
73 2019-11-29 23:13:28,057 [Thread-2 ] [INFO ] Notification reveived!
74 2019-11-29 23:13:28,059 [Thread-2 ] [INFO ] Single notification received
75 2019-11-29 23:13:28,063 [Thread-2 ] [INFO ] MQTT log:Sending PUBLISH (d0, q0, r0, m1),
    ↳ 'b'/ble/f17a1fe9d42b711c463cdad54e5db6f0/readValue'', ... (3 bytes)
76 2019-11-29 23:13:28,064 [Thread-2 ] [INFO ] Waiting for further instructions...
77 2019-11-29 23:14:15,127 [Thread-1 ] [INFO ] MQTT log:Sending PINGREQ
78 2019-11-29 23:14:15,129 [Thread-1 ] [INFO ] MQTT log:Received PINGRESP

```


Client debug log

```

1 2019-11-29 23:13:25,338 [MainThread ] [INFO ] Loading client key pair...
2 2019-11-29 23:13:25,366 [MainThread ] [INFO ] Starting MQTT client
3 2019-11-29 23:13:25,367 [MainThread ] [INFO ] Fetching access ticket from A3C GW server
  ↪ c81306517f41a51b92445538e3406296
4 2019-11-29 23:13:25,372 [MainThread ] [INFO ] Successfully fetched access ticket from A3C
  ↪ GW server c81306517f41a51b92445538e3406296
5 2019-11-29 23:13:25,373 [Thread-1 ] [INFO ] Starting Client <-> GW session. GW UUID
  ↪ 50433dbaa1de8cf4c381302970dd7f7f
6 2019-11-29 23:13:25,480 [Thread-1 ] [INFO ] Successfully established session with GW
  ↪ UUID: 50433dbaa1de8cf4c381302970dd7f7f
7 2019-11-29 23:13:25,480 [Thread-1 ] [INFO ] Fetching access ticket from A3C DD server
  ↪ EE3AEEDD1BC30C4D2472D638064151C4
8 2019-11-29 23:13:25,500 [Thread-1 ] [INFO ] Successfully fetched access ticket from A3C
  ↪ DD server EE3AEEDD1BC30C4D2472D638064151C4
9 2019-11-29 23:13:27,158 [Thread-1 ] [INFO ] Received decrypted nonce matches session
  ↪ nonce
10 2019-11-29 23:13:27,182 [Thread-1 ] [INFO ] MQTT log: Sending CONNECT (u0, p0, wr0, wq0,
  ↪ wf0, c1, k60) client_id=b''
11 2019-11-29 23:13:27,184 [Thread-1 ] [INFO ] MQTT log: Sending SUBSCRIBE (d0, m1)
  ↪ [(b'/ble/f17a1fe9d42b711c463cdad54e5db6f0/readValue', 0)]
12 2019-11-29 23:13:27,185 [Thread-2 ] [INFO ] MQTT log: Received CONNACK (0, 0)
13 2019-11-29 23:13:27,186 [Thread-2 ] [INFO ] MQTT: New connection established
14 2019-11-29 23:13:27,188 [Thread-2 ] [INFO ] MQTT log: Received SUBACK
15 2019-11-29 23:13:28,078 [Thread-2 ] [INFO ] MQTT log: Received PUBLISH (d0, q0, r0, m0),
  ↪ '/ble/f17a1fe9d42b711c463cdad54e5db6f0/readValue', ... (3 bytes)
16 2019-11-29 23:13:28,078 [Thread-2 ] [INFO ] MQTT: New message received on topic
  ↪ /ble/f17a1fe9d42b711c463cdad54e5db6f0/readValue Payload=b'123'
17 2019-11-29 23:14:28,144 [Thread-2 ] [INFO ] MQTT log: Sending PINGREQ
18 2019-11-29 23:14:28,147 [Thread-2 ] [INFO ] MQTT log: Received PINGRESP

```


DH internal states

State name	Value	Description
DH_AUTH_REQ	0x10	DH has initialized successfully and is currently waiting to create a secure session with a Gateway
DH_AUTH_NONCE_EXC_REQ	0x20	DH has successfully validated the ticket and is currently waiting to receive the encrypted nonce in order to finalize the session establishment
DH_AUTH_OK	0x30	DH has successfully created a secure session with a Gateway
DH_PUB_KEY_LOAD_ERROR	0xF0	DH failed while reading its public key from memory
DH_PRIV_KEY_LOAD_ERROR	0xF1	DH failed while reading its private key from memory
DH_PART_MOUNT_ERROR	0xF2	DH failed while mounting its local partition
DH_ERROR	0xFF	For generic error handling

Table H.1: DH internal states.