Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

MASTER'S DEGREE IN TELECOMMUNICATIONS ENGINEERING

MASTER'S THESIS

IN-NETWORK VALIDATION OF DIGITAL CERTIFICATES FOR IoT SECURE COMMUNICATIONS

Student	Zufiaurre Soto, Gloria
Director	Astorga Burgo, Jasone
Co-director	Jacob Taquet, Eduardo
Departament	Communications Engineering
Academic Year	2020/2021

Bilbao, 21 September 2021

ABSTRACT

This project provides a design and implementation of a certificate validation system for Internet of Things (IoT) scenarios. During the establishment of a secure communication, a significant step to carry out by an endpoint is to authenticate the remote communication peer, which usually involves validating the digital certificate of that peer. Unfortunately, this leads to a significant computing cost; thus, validation becomes difficult for IoT devices to perform. Therefore, this project proposes a solution whose scope is to delegate certificate validation actions to another intermediate device in the network in a totally transparent way for the remote peers. In this case, a programmable switch is going to check the authenticity of the endpoint an IoT device is going to communicate with. In particular, it will verify the signature of the certificate and check whether it is revoked.

This system will be implemented on the basis of Software Defined Networking (SDN) and In-Network Computing (INC) as they are particularly well-suited to IoT scenarios. In order to make the switch carry out validation operations, both its control and data plane will be programmed. Finally, the impact that this solution has on network performance will be evaluated.

Key words: certificates, IoT, validation, programmable switch, security, SDN, INC

LABURPENA

Proiektu honek Internet of Things-en (IoT) inguruneetan ziurtagiriak balioztatzen dituen sistema baten diseinua eta inplementazioa aurkezten ditu. Komunikazio seguru bat ezarri arte ematen den prozesuan ezinbestekoa da komunikazio amaierako entitateak beste muturra balioztatzea, honek dakarren ziurtagiri digitalaren balioztapenarekin batera. Zoritxarrez, honek balio konputazional handia du, eta beraz, zaila da IoT gailuetan balioztapena egitea. Horregatik, proiektu honek hurrengo helburua duen soluzio bat proposatzen du: ziurtagiria balioztatzeko ekintzak sareko bitarteko beste gailu baten esku uztea, modu guztiz gardenean. Kasu honetan, IoT gailuarekin komunikatuko den urruneko muturreko baliozkotasuna switch programagarri baten bidez frogatuko da. Bereziki, ziurtagiriaren sinadura baieztatuko du eta baliogabetua dagoen frogatuko du.

Sistema hau software bidez definitutako sareetan (SDN) eta sareko konputazioan (INC) oinarrituko da. Izan ere, IoT inguruneetara bereziki ongi egokitzen dira. Switch-ak balioztatze ekintzak aurrera eramateko, bere kontrol planoa zein datuen planoa programatuko dira. Azkenik, soluzio honek sarearen errendimenduan duen eragina ebaluatuko da.

Gako-hitzak: ziurtagiriak, IoT, baliozkotasuna, switch programagarria, segurtasuna, SDN, INC

RESUMEN

Este proyecto proporciona un diseño e implementación de un sistema de validación de certificados para escenarios de Internet of Things (IoT). En el transcurso del establecimiento de una conexión segura, una acción importante que debe llevar a cabo una entidad final de la comunicación es autenticar el otro extremo, lo que normalmente implica validar el certificado digital de dicho extremo. Desafortunadamente, esto conlleva un coste computacional importante, por lo que la validación resulta difícil de realizar para los dispositivos IoT. Por ello, este proyecto propone una solución cuyo objetivo es delegar las acciones de validación del certificado a otro dispositivo intermedio de la red de forma totalmente transparente para los extremos. En este caso, un switch programable va a comprobar la autenticidad del extremo remoto con el que se va a comunicar un dispositivo IoT. En particular, verificará la firma del certificado y comprobará si este está revocado.

Este sistema se implementará basándose en las redes definidas por software (SDN) y la computación en red (INC), ya que se adaptan especialmente bien a escenarios IoT. Para que el switch realice las operaciones de validación, se programará tanto su plano de control como el de datos. Por último, se evaluará el impacto que esta solución tiene sobre el rendimiento de la red.

Palabras clave: certificados, IoT, validación, switch programable, seguridad, SDN, INC

TABLE OF CONTENTS

ABSTRACT **2**

LABURPENA **3**

RESUMEN **4**

LIST OF ACRONYMS **12**

1 INTRODUCTION **15**

2 BACKGROUND **17**

2.1 INTERNET OF THINGS - IoT 17

2.1.1 IoT system architecture *17*

2.1.2 IoT platform *19*

2.1.3 IoT security issues *19*

2.2 DATAGRAM TRANSPORT LAYER SECURITY - DTLS 21

2.2.1 DTLS handshake protocol *23*

2.3 PUBLIC KEY INFRASTRUCTURE 25

2.3.1 Digital certificates. *25*

2.3.2 Certificate Authority -- CA *27*

2.4 SOFTWARE DEFINED NETWORKING – SDN 30

2.4.1 Architecture *31*

2.4.2 OpenFlow *32*

2.4.3 SDN in IoT scenarios *34*

2.5 IN-NETWORK COMPUTING – INC 35

2.5.1 Data Plane programming *35*

3 OBJECTIVES **40**

3.1 MAIN OBJECTIVE 40

3.2 SECONDARY OBJECTIVES 40

3.2.1 Architecture design *40*

3.2.2 Study of the available tools and selection of the best alternatives for later implementation *40*

3.2.3 Network model implementation *41*

3.2.4 Validation of the implemented solution *41*

3.2.5 Analysis of results and conclusions drawing *41*

4 BENEFITS **42**

4.1 TECHNICAL BENEFITS 42

4.2 ECONOMIC BENEFITS 43

4.3 SOCIAL BENEFITS 43
5 REQUIREMENTS 45
6 STATE OF THE ART 46
7 ALTERNATIVES ANALYSIS 48
7.1 PROGRAMMABLE SWITCH SOFTWARE 48
Mininet *48*
7.1.1 P4-lang tutorials VM *..... 49*
7.1.2 P4-utils VM *..... 50*
7.2 IMPLEMENTATION OF DTLS HANDSHAKE 52
7.2.1 Scandium -- Californium *..... 52*
7.2.2 Python3-DTLS *..... 53*
7.3 Certificate Validation libraries 56
7.3.1 pyOpenSSL *..... 56*
7.3.2 cryptography *..... 57*
7.4 CA and OCSP responder 58
7.4.1 XiPKI *..... 58*
7.4.2 Enterprise Java Beans Certificate Authority - EJBCA *..... 59*
8 RISK ANALYSIS 63
8.1 RISK DESCRIPTION 63
8.1.1 R1: Unavailability of project members *..... 63*
8.1.2 R2: Planification delays *..... 63*
8.1.3 R3: Data loss or equipment failure *..... 64*
8.1.4 R4: Previously undetected software incompatibility *..... 64*
8.2 RISK PROBABILITY-IMPACT MATRIX 64
8.3 CONTINGENCY MEASURES 65
8.3.1 R1: Unavailability of project members *..... 65*
8.3.2 R2: Planification delays *..... 65*
8.3.3 R3: Data loss or equipment failure *..... 65*
8.3.4 R4: Previously undetected software incompatibility *..... 66*
9 SOLUTION DESIGN AND IMPLEMENTATION 67
9.1 DESIGN 67
9.2 IMPLEMENTATION 69
9.2.1 Mininet Network deployment *..... 70*
9.2.2 H4: Certificate Authority, OCSP responder – EJBCA *..... 72*

9.2.3 H1 & H2: DTLS Handshake – Python3, OpenSSL ... *76*
9.2.4 S1: Switch data plane – P4 ... *78*
9.2.5 H3: Controller – Python3 libraries ... *83*
9.2.6 DTLS handshake start-up ... *85*
10 ANALYSIS OF RESULTS ... 90
10.1 FUNCTIONAL VERIFICATION TEST – FVT ... 90
10.1.1 FVT1 -- OCSP server ... *90*
10.1.2 FVT2 – Traffic analysis ... *91*
10.2 SYSTEM PERFORMANCE ... 93
10.2.1 Case 1: No fragmentation – Validation at the controller side ... *94*
10.2.2 Case 2: No fragmentation – Validation at client side ... *94*
10.2.3 Case 3: Fragmentation – Validation at the controller side ... *95*
10.2.4 Case 4: Fragmentation – Validation at the client side ... *96*
10.2.5 Comparison of the results ... *96*
11 WORK METHODOLOGY ... 100
11.1 WORKING TEAM ... 100
11.2 WORK PACKAGES ... 100
11.2.1 WP1: Problem statement ... *100*
11.2.2 WP2: Solution design and working environment installation ... *101*
11.2.3 WP3: Implementation of a basic network ... *101*
11.2.4 WP4: Implementation of the entire network architecture ... *102*
11.2.5 WP5: Solution validation ... *102*
11.2.6 WP6: Project management and documentation ... *103*
11.3 GANTT DIAGRAM ... 104
12 ECONOMIC COSTS ... 105
12.1 HUMAN RESOURCES ... 105
12.2 DEPRECIATION COSTS ... 105
12.3 OTHER COSTS ... 106
12.4 TOTAL COSTS ... 106
13 CONCLUSIONS AND FUTURE WORK ... 107
14 REFERENCES ... 108
ANNEXES ... 111
Annex I: Switch's data plane: P4 program ... 111
Annex II: Switch's control plane: Controller python script ... 119

Annex III: Switch's configuration files: 122
Annex IV: EJBCA setup configuration file: 124

LIST OF FIGURES

Figure 1: Number of IoT connected devices worldwide from 2019 to 2030 (in billions) [2] 15
Figure 2: IoT architecture layers..... 18
Figure 3: Time it takes for a hacker to crack a password..... 20
Figure 4: Echanged messages during DTLS handshake..... 23
Figure 5: X.509v3 certificate structure..... 26
Figure 6: Top security concerns a survey participants voiced related to the widespread use of digital certificates [18]..... 27
Figure 7: OCSP checking architecture 29
Figure 8: Comparison between traditional network architecture and SDN architecture..... 30
Figure 9: Software Defined Networking architecture..... 31
Figure 10: OpenFlow architecture 33
Figure 11: Examples of OpenFlow table entries 34
Figure 12: IoT integration in SDN 35
Figure 13: Data plane programming 37
Figure 14: Target-independent vs. Architecture-dependent..... 37
Figure 15: P4 V1model architecture 38
Figure 16: P4 target..... 39
Figure 17: Mininet vs. traditional network 48
Figure 18: Exchanged message during DTLS handshake using Scandium repository..... 52
Figure 19: DTLS handshake Certificate message..... 53
Figure 20: Exchanged message during DTLS handshake using Python3-DTLS repository 54
Figure 21: DTLS handshake Certificate message..... 54
Figure 22: OCSP responder in EJBCA..... 60
Figure 23: IoT scenario project design 67
Figure 24: Network model 69
Figure 25: Parser states..... 80
Figure 26: Ingress match-action pipeline state diagram..... 81
Figure 27: Egress match-action pipeline state diagram..... 82
Figure 28: Controller actions state diagram..... 84
Figure 29: DTLS handshake first stage 85
Figure 30: DTLS handshake second stage 86

Figure 31: DTLS handshake third stage 87
Figure 32: Last stage of the DTLS handshake 88
Figure 33: Traffic flow between DTLS server and switch 91
Figure 34: Traffic Flow between DTLS client and switch 92
Figure 35: Traffic flow between controller and switch data plane. 92
Figure 36: Traffic flow between OCSP responder and switch 92
Figure 37: Mean of certificate revocation checking time of each evaluated case 96
Figure 38: Mean of certificate validation time of each evaluated case 97
Figure 39: Mean of Certificates handling time of each evaluated case 98
Figure 40: Mean of Certificates handling time of each evaluated case 98

LIST OF TABLES

Table 1: Evaluation of programable switch software 51
Table 2: Evaluation of DTLS handshake repositories 56
Table 3: Evaluation python libraries 57
Table 4: Evaluation of software that provides OCSP and CA services 62
Table 5: Risk probability-impact Matrix 64
Table 6: Hosts' configuration 71
Table 7: Switch match-action rules 72
Table 8: Time measurements in milliseconds, controller validation, no fragmentation 94
Table 9: Time measurements in milliseconds, client validation, no fragmentation 94
Table 10: Time measurements in milliseconds, controller validation, fragmentation 95
Table 11: Time measurements in milliseconds, controller validation, fragmentation 96
Table 12: Working team 100
Table 13: Gannt diagram of the project 104
Table 14: Costs of human resources 105
Table 15: Depreciation costs 105
Table 16: Other costs 106
Table 17: Total costs 106

LIST OF ACRONYMS

ARP	Address Resolution Protocol
API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
CA	Certificate Authority
CDP	CRL Distribution Point
CPU	Central Processing Unit
CRL	Certificate Revocation Lists
CSR	Certificate Signing Request
DAS	Data Acquisition System
DB	Database
DH	Diffie-Hellman
DNS	Domain Name System
DDoS	Distributed Denial of Service
DTLS	Datagram Transport Layer Security
EdDSA	Edwards-curve Digital Signature Algorithm
EJBCA	Enterprise Java Beans Certificate Authority
ESP	Encrypting Security Payload
FPGA	Field Programmable Gate Array
FVT	Functional Verification Test
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IEEE	Institute of Electrical and Electronics Engineers

IETF	Internet Engineering Task Force
INC	In-Network Computing
IoT	Internet of Things
IP	Internet Protocol
IT	Information Technology
LAN	Local Area Network
M2M	Machine to Machine
MAC	Message Authentication Code
MTU	Maximum Transmission Unit
NAT	Network Address Translation
NPU	Neural Processing Unit
OCSP	Online Certificate Status Protocol
ONF	Open Networking Foundation
OWASP	Open Web Application Security Project
PISA	Protocol-Independent Switch Architecture
PKCS	Public Key Cryptography Standard
QoS	Quality of Service
RA	Registration Authority
RAM	Random Access Memory
SDN	Software Defined Networking
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol

VA	Validation Authority
VM	Virtual Machine
VPN	Virtual Private Network

1 INTRODUCTION

Internet security is a broad issue which is on the rise. In terms of securing personal devices, there is a tendency to focus on computers, tablets or smartphones, as they are the most daily used electronic devices. Nevertheless, there are many more connected devices to the Internet, better known as Internet of Things (IoT). In a nutshell, IoT addresses a system of interrelated and internet-connected objects which collect and send data over a wireless network without human intervention [1]. E.g., smart fire alarms, industrial or medical sensors among others. These devices collect more information than one might imagine, hence, the need of securing them is highly important. Unfortunately, security in IoT is not in the public eye as much as smartphones' or computers' security may be. In recent years, there has been a large increase in the number of IoT devices due the emergence of new technologies and the evolution of all sectors in society towards the digitization of services. Figure 1 shows the number of IoT connected devices worldwide and an estimation of this number in the coming years.

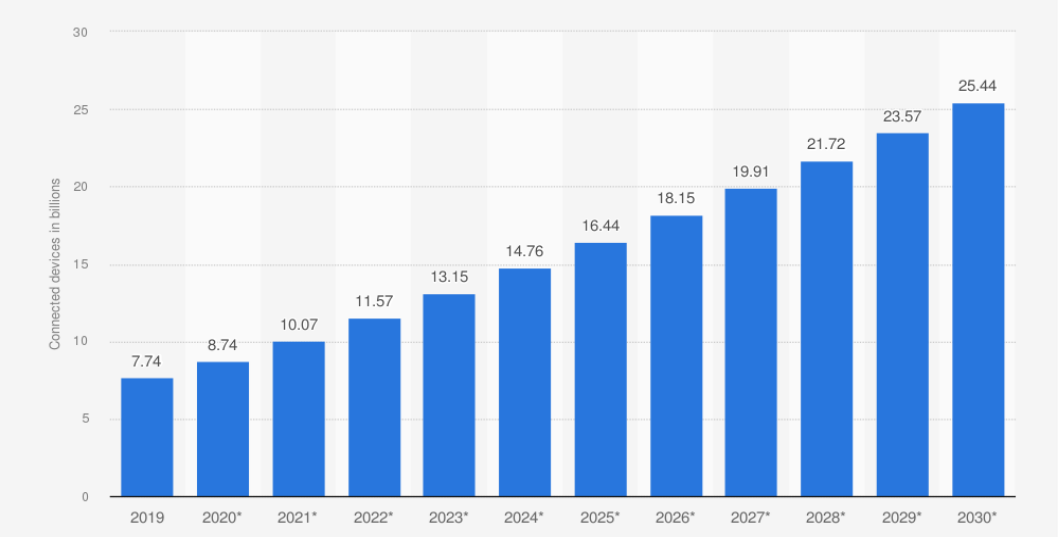

Figure 1: Number of IoT connected devices worldwide from 2019 to 2030 (in billions) [2]

In terms of secure communications, Public Key Infrastructure (PKI) [3] is a cornerstone of it. It is intended both to manage public key encryption and to handle digital certificates. Firstly, public key or asymmetric encryption, is a method of encrypting data that enables establishment of two different shared symmetric keys between parties that have never met. However, asymmetric cryptography alone is not enough to solve the key distribution problem, additional security is needed. There must be an entity responsible for claiming that a given key belongs to a subject, a CA [4]. It issues digital certificates which certify whom a given public key belongs to. These certificates let websites, clients and devices verify they are who they claim to be, in other words, authenticate. Hence, entities which take part in a communication can trust each other and establish keys in order to encrypt and decrypt transferred data.

The most common way of securing Internet communications is via Transport Layer Security (TLS) which is the basis that HTTP uses in order to make HTTPS possible. The equivalent protocol to

TLS for communications over UDP, such as IoT communications, is Datagram Transport Layer Security (DTLS). Both asymmetric encryption and CA are important building blocks of DTLS [5]. DTLS is used to secure data sent over an untrusted network and is particularly appropriate for securing IoT applications. DTLS consists of some subprotocols, one of which is a handshake protocol [5]. This one allows the server to authenticate and negotiate a shared symmetric key with the client. The server sends its digital certificate to the client, who contacts a Certification Authority to validate it. Nevertheless, this validation implies a significant computing and communication cost.

Taking all this into account, the need of using valid certificates is pretty important. Even though every certificate has its validity period, they must sometimes be revoked before expiring. A certificate revocation is mainly performed when an encryption key has been compromised or the user is no more certified by the same CA, among other reasons. Whenever an IoT device receives a server's certificate, it must verify its signature and must also take some actions in order to ensure that the received certificate is not revoked. Unfortunately, as already mentioned, these operations imply a significant cost and are not suitable to be performed on IoT devices, which are poor in resources. This leads to some critical issues; an adversary could have substituted the server's public key and it establishes a secure session with the IoT device making the client believe it is communicating with a trustworthy server instead of an adversary, better known as a Man in The Middle attack [6].

On the one hand, Software Defined Networking (SDN) is a new technology that consists of abstracting network functions in order to virtualize or control them by software. It separates the network's control and forwarding planes; thus, a centralized view of the distributed network is provided. Therefore, it allows network administrators to adapt a network to the different dynamic applications demanded requirements such as service velocity or customized network operations. On one side, a centralized controller can be responsible of implementing all the control functions of a network such as routing and forwarding decisions. On the other side, some SDN elements allow to program its data plane, which lets them offer personalised functions.

On the other hand, In-Network Computing (INC) is an emergent technology that refers to the execution of programs which usually run on end-hosts within network devices. Traditional network devices are fixed-function and support only defined functionalities, whereas programmable network devices let users implement their functionalities. No extra space, cost or idle power is required when using INC as switches and Network Interface Card (NIC) are already used for networking functions, the sole difference is that their functionalities are just extended.

Looking into the existing technologies and the problems regarding IoT security, the aim of this master thesis is to delegate digital certificates validation actions, which should be carried out by an IoT device, to a network infrastructure entity such as a programmable switch. It will be based on SDN and INC as they are very well suited to IoT scenarios. Thus, this project will develop a system which validates digital certificates on behalf of IoT devices but outside them.

2 BACKGROUND

In order to contextualize this project, a brief overview of the concepts and technologies this project is based on is provided: Internet of Things, Public Key Infrastructure, Datagram Transport Layer Security, Software Defined Networking and In-Network Computing.

2.1 INTERNET OF THINGS - IoT

IoT addresses the interconnection of devices and objects through a network (either private or the Internet) where all devices are visible and can interact with each other. An IoT device can be anything from sensors and mechanical devices to everyday objects such as smart refrigerators, or watches. In a nutshell, anything that can be connected to the internet with no need of human action, better known as Machine to Machine (M2M) interaction.

A device is considered as an IoT one when it is capable of connecting to the Internet and presents some other technology such as sensors, functional software that allows it to communicate with both networks and actuators.

2.1.1 IoT system architecture

It must be mentioned that there are no recognized standards for IoT network architecture. Basing on data flows and functionalities, it can be often described as a four stage/layer process architecture that represents how data is sent through a network between sensors and a data centre or the cloud. Figure 2 shows these layers. It may be the case that a sensor, for instance, gets information and sends it to the data centre, where it is finally processed, analysed, and stored or it can also happen in the other direction, an actuator can receive instructions or commands from a data centre that tell it to take some action to control a physical process. This process may be something like turning on a light or shutting down an assembly line whenever a failure occurs. Turning to the subject of four stages-based architecture, a description of each is shown below.

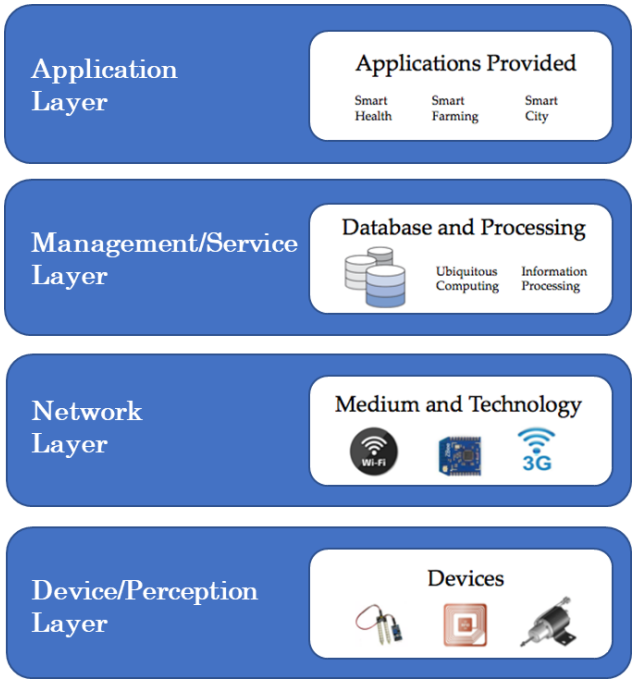

Figure 2: IoT architecture layers

- **Perception/device layer:** This stage addresses all connected devices (sensors and actuators basically) which are responsible for monitoring or controlling a physical process, a machine, a building or even a person. The main difference between a sensor and an actuator is that the sensor monitors the physical process, whereas an actuator controls it. On the one hand, data collected by sensors is usually about the status of a process or an environmental state such as temperature or chemical composition among many others. Sensors and actuators may work together, a sensor can detect an environmental condition that needs a quick response an actuator can carry out.

- **Network layer:** This layer consists of Internet gateways and Data Acquisition Systems (DAS). DAS function is to convert from analog to digital format all the data received from sensors. Afterwards, it sends it through an Internet gateway both over a Wired network or a Wireless one.

- **Management/service layer:** At this stage, due to the big amount of digital presented data a processing action must be applied to it so as to reduce the data volume. Thus, it will be possible to send data to a data centre or a cloud as it takes up less space. This processing is usually performed by an edge device, closed to sensors or actuators in order to process it.

- **Application layer:** Last layer contains powerful systems such as a data centre or the Cloud. They analyse data in depth, manage it and store it in a secure way. Their main task is to join data obtained from different sensors in order to generate a broad overview of the IoT system. Then, this overview is provided to IT and business managers. It must be highlighted that data is also stored in a data stock so as to record it and to

keep it in case further analysis is needed. At this point, applications decide if an action should be taken and which one exactly.

2.1.2 IoT platform

IoT needs software to be able to operate, involving middleware better known as IoT platform [7]. This platform is placed between the layers of IoT devices and IoT gateways and applications. It is essential to bridge between the layers and data network. In a nutshell, an IoT platform makes IoT easier for developers, users and businesses to handle. For instance, developers can easily collect remote data, create and distribute applications, manage and ensure connectivity of devices. An IoT platform consists of a set of components, connected with each other, which guarantee a continuous communication flow between devices. Its structure classifies traffic and defines which type of received data is sent where and how much processing is carried out at each layer. It is strongly recommended for companies to use IoT platforms, since it provides all the common functionalities for an application, thus, one can focus on creating specific features of a product that make it different from others at the market.

2.1.3 IoT security issues

IoT has not fully evolved at all as far as security is concerned, which means that IoT devices are most likely to be vulnerable to attacks and therefore suffer from data leakages. There are several IoT security risks, OWASP (Open Web Application Security Project) [8] has considered the following risks as the most critical ones.

1. **Weak, guessable, or hardcoded passwords**. There is usually a tendency to use short or simple passwords which let an adversary guess them quicklier than one can even imagine. Devices sometimes do not allow users to change the default password or users just do not want to change it. Moreover, success in accessing one device usually grants the adversary access to others in the system since IoT devices often have the same default password. There are several online applications such as *howsecureismypassword* [9] which tells how secure a password is, based on the information provided in figure 3. Furthermore, in order to know deeper how hackers usually hack passwords, Dale Walker tells the top 12 password-cracking techniques used by hackers [10].

Number of Characters	Numbers Only	Lowercase Letters	Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters, Symbols
4	Instantly	Instantly	Instantly	Instantly	Instantly
5	Instantly	Instantly	Instantly	Instantly	Instantly
6	Instantly	Instantly	Instantly	1 sec	5 secs
7	Instantly	Instantly	25 secs	1 min	6 mins
8	Instantly	5 secs	22 mins	1 hour	8 hours
9	Instantly	2 mins	19 hours	3 days	3 weeks
10	Instantly	58 mins	1 month	7 months	5 years
11	2 secs	1 day	5 years	41 years	400 years
12	25 secs	3 weeks	300 years	2k years	34k years
13	4 mins	1 year	16k years	100k years	2m years
14	41 mins	51 years	800k years	9m years	200m years
15	6 hours	1k years	43m years	600m years	15 bn years
16	2 days	34k years	2bn years	37bn years	1tn years
17	4 weeks	800k years	100bn years	2tn years	93tn years
18	9 months	23m years	6tn years	100 tn years	7qd years

Figure 3: Time it takes for a hacker to crack a password

2. **Insecure network services:** If there are services running on an IoT device which do not present proper security, whenever they are exposed to the internet they are also exposed to remote unauthorized access and so, data leakage. Then, attackers can compromise the IoT device.

3. **Insecure ecosystem interfaces:** The web and mobile interface, the backend API and the cloud are interfaces that let the user interact with the device. Hence an adversary can compromise a device throughout these interfaces if they are not secure enough. It is fairly common to encounter vulnerabilities in them such as weak encryption, lack of authentication or authorization or not filtering data.

4. **Lack of secure update mechanisms:** A device is not able to update itself in a secure way. Thus, they are likely to be short of firmware validation, anti-rollback mechanisms, security update notifications and secure delivery.

5. **Use of Insecure or Outdated components:** The use of deprecated or insecure software or hardware could lead to a total compromise of the device. For instance, the use of expired or revoked digital certificates is a security breach since an adversary could be using a revoked certificate which was revoked due to compromising its key. As the IoT device is using an outdated certificate, it is not updated about its revocation state, it believes the entity it is communicating with is who the certificate claims to be and not an adversary. Hence, it is really important to have hardware and software components updated.

6. **Insufficient privacy protection:** IoT devices use and store sensitive and considerable information about the environment they are in and people using them. Unfortunately, these devices do not typically offer secure storage neither of personal information nor of the environment they are being used, which leads to data leakage.

7. **Insecure data transfer and storage:** The lack of data encryption in transit, processing or at rest tempts hackers to steal and expose data. To mitigate this risk, it is pretty convenient to restrict access to sensitive data and make sure that data is always encrypted.

8. **Lack of device management:** There is a lack of ability to secure in an effective way all the devices on the network. They tend to be short of asset and update management, secure decommissioning, monitoring systems and response capabilities. Hence, the whole system is exposed to threats.

9. **Insecure default settings:** IoT devices are often delivered with insecure default settings. Sometimes system configurations are all immutable or most of them restricted or they can be changed, but it is the client who may be careless and fail to change them. For example, one can encounter with fixed passwords or inability to keep up to date with security updates amongst many others.

10. **Lack of physical hardening:** It is significant to harden a device against physical attacks. a security breach at this point will let attackers access to sensitive data that can lead to get local control of the device or throw a remote attack.

Thanks to this OWASP consideration, business and customers know which kind of vulnerabilities their IoT devices are exposed to and avoid unpleasant surprises due to lack of knowledge. Prateek Panda [11] explains how one can alleviate each of the OWASP IoT risks. Nevertheless, a single solution cannot address all of them, the use of more than one is needed. The proposed solution in this project is focused on mitigating *Use of Insecure or Outdate components*.

As already explained, the use of insecure or outdate components in software can lead to use invalid certificates. In the case of an IoT scenario, it would be critical that the remote peer the IoT device wants to communicate with uses invalid certificates. It is significant to know whether a certificate is revoked or not to avoid attacks like the following that a Java application suffered from [12] due to not checking revocation status of certificates. As the adversary was using a revoked certificate due to a key compromise, however much it is revoked, if the client does not know about it, it is useless.

2.2 DATAGRAM TRANSPORT LAYER SECURITY - DTLS

DTLS provides security to datagram-based protocols such as UDP. Some main facts of UDP are mentioned below so as to understand the role DTLS plays.

UDP is a transport layer level protocol based on datagram transmission. It allows communication with no need for any agreement between parties before transferring data since it gives more importance to speed than to reliability. Moreover, it does not order packets, as the most common protocol used on the Internet does, Transmission Control Protocol (TCP). Another significant fact is that UDP does not guarantee that a packet has already arrived at its destination. Thus, UDP is particularly used for communicating between time-sensitive

applications on the internet which demand low-latency and loss-tolerating. Then, one must know that if UDP is being used by its application, data integrity is not either considered.

UDP is typically chosen by IoT manufacturers since it demands few network resources and does not need to keep a persistent connection between communication parties. Hence, UDP adapts to IoT application requirements. IoT devices do not send neither confirmation of arrived packets nor retransmission, thus, the required downlink budget is rather low. Furthermore, losing a single datapoint in IoT is not a matter of the very greatest importance as the device sends data periodically. The device can turn off after sending or receiving data. Hence UDP is a helpful protocol in this scenario. Having understood the main aspects of UDP and its satisfaction of IoT requirements, DTLS can be explained.

As already mentioned at the beginning of this section, DTLS provides security to datagram-based protocols. In addition, it allows applications to communicate in such a way that they prevent eavesdropping [13] and tampering, better known as man in the middle attack. It is based on Transport Layer Security (TLS) protocol, which provides security mostly to computer-based communications. They differ from each other in the protocol they go over, TLS uses TCP, whereas DTLS goes over UDP. TLS cannot be used over UDP since UDP can tolerate packet loss and TLS does not have internal facilities to handle unreliability. The aim of DTLS is to make slight changes to TLS so as to face this problem.

Independent decryption of packets is not possible with TLS, since it uses implicit sequence numbers to check integrity, whereas DTLS uses specific sequence numbers to solve this problem. Furthermore, TLS ensures messages reception reliability, thus, a connection is closed if one message is lost. DTLS solves this using a retransmission timer for packets.

It is worth mentioning that TLS and DTLS use both asymmetric and symmetric encryption to ensure confidentiality and integrity of exchange data. Asymmetric encryption is used to establish a secure session between a client and a server, whereas symmetric encryption is used to exchange data through the secure session. In a nutshell, asymmetric encryption is a cryptography system that uses pairs of keys. The pair consists of a public key, which can be access by anybody, and a private key, which is kept in private by the owner. Any of those keys can either be used for encrypting or decrypting and anything encrypted with one key can only be decrypted with other and vice versa.

Both TLS and DTLS are composed of different subprotocols, the main ones being handshake protocol and transport protocol. The first one is used for the establishment of the secure connection, in a nutshell, it allows both the server and the client to authenticate and negotiate a shared symmetric key. The second one, transport protocol, is the one that actually encrypts and authenticates user information. A description of DTLS handshake protocol is provided below since it is responsible for authenticating the communications end-entities, which is usually done through the exchange and validation of digital certificates.

2.2.1 DTLS handshake protocol

DTLS handshake is carried out between the client and the server of a communication to authenticate each other and negotiate the encryption algorithms they will use, and so, agree on session keys in order to use symmetric encryption. DTLS messages are almost identical to those of TLS handshake with some slight differences.

- As an adversary can use too many resources on the server by sending several handshake initiations requests, a stateless cookie exchange is introduced to mitigate DoS (Denial of Service) attacks. When the client sends a *ClientHello* message, the server now answers with a *HelloVerifyRequest* message which contains a stateless cookie. After the client has received this message, it must answer back with a new *ClientHello* message including that cookie. Once the server verifies the cookie, it goes on with the handshake. This makes it hard for an adversary with a spoofed IP address to commit the DoS attack.
- The handshake header is also modified so as to cope with message loss and reordering. As DTLS handshake messages are considerably larger than any other datagram, DTLS message fragmentation must also be handled in the header, since IP fragmentation must be avoided as far as possible.
- In addition, retransmission timers are also added to handle message loss.

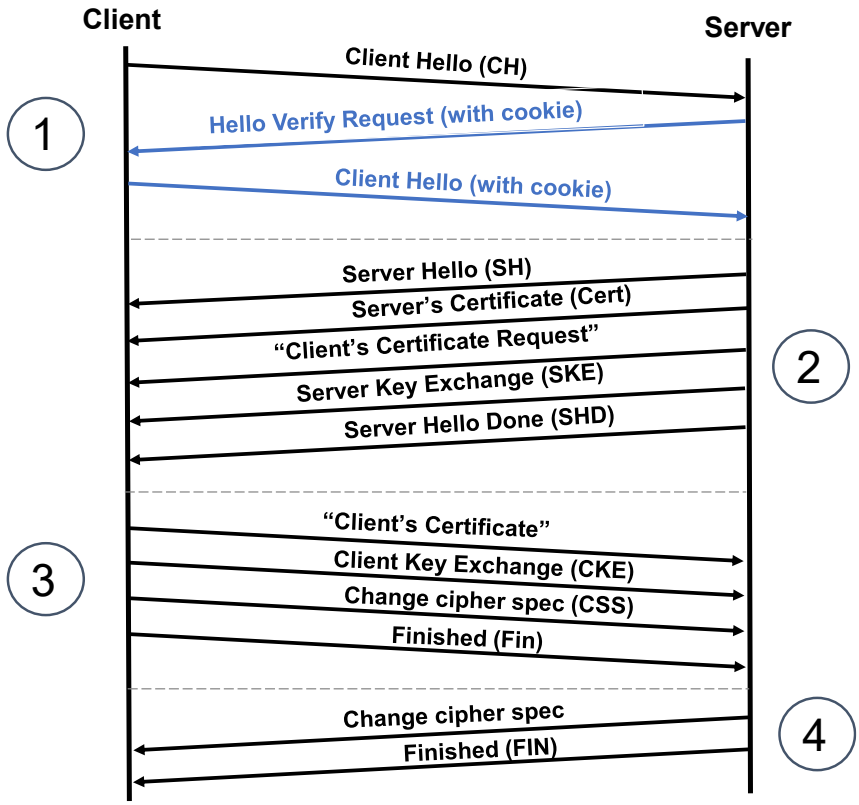

Figure 4: Echanged messages during DTLS handshake

Figure 4 shows exchange messages during a DTLS handshake. The steps followed in handshake are described below:

1. DTLS handshake is initiated by the client. First, it sends to the edge server a *ClientHello* message which contains a random value and a list of cryptographic methods supported by the client. The server responds with a *HelloVerifyRequest* which contains the previously mentioned cookie. DTLS client answers with a second *ClientHello* message containing the cookie.

2. After the server receives the second *ClientHello* message, it checks the received cookie, if it is correct, it will answer back with a *ServerHello* message. This message indicates a choice of the encryption options previously provided by the client. It is followed by another message containing the server's digital certificate so as to prove authenticity to the client. After *certificate* message, *ServerKeyExchange* is sent, which contains authenticated information about the key exchange. It must be mentioned that the client already has this information, which consists of the following: received *ClientHello* message with its random value, the already sent *ServerHello* message with its random value as well as parameters of the Diffie-Hellman algorithm used to create the key pair. That information is joined together so as to create a message (m), which is finally signed with the server's private key. This signed message is sent to the client.

3. As already mentioned, the client has the information used to create the message, which is later signed, then, it creates the same message as the server did joining that information (m'). Furthermore, the client gets the server's public key from the received certificate. With that public key it is capable of decrypting the message that the server signed with its private key. Thus, it can compare if the decrypted message (m) is the same as the one it has just created (m'). In a nutshell, the client verifies if the signature of the SKE message was done by the entity of the received certificate.

   This is a considerable step since the client verifies the received certificate. It checks if it is issued by the CA the certificate indicates and it also checks its revocations state. It is significant to check if a certificate has been revoked since one reason why a certificate has been revoked is due to a compromised key, then, using it will lead to serious problems. Alternatively, the server can send other messages such as asking the client for its certificate. Hence, a *ServerHelloDone* message is required at the end of this step so as to know where the server has finished.

4. Once the client has received *ServerHelloDone* message, it knows that the server has sent over all its messages. If requested by the server, the client sends its certificate. At this point, the client is capable of generating session keys. Depending on the key exchange method agreed on the initial *ClientHello* and *ServerHello* messages, the client figures out a pre-master secret, which is a random string of bytes. It encrypts this secret with the server's public key and sends it to the server through the *ClientKeyExchange* message. It is also sent a covering message, *ChangeCipherSpec* which lets the server know the client has generated the session key and that the following sent messages will be

encrypted. Finally, the client sends a *Finished* message, which means the handshake is finished from the client side. This last message is the first one which is encrypted, then, it is the first sent information protected by the session key. It also contains a MAC (Message Authentication Code) which guarantees that the handshake has not been tampered with.

5. Eventually, the server decrypts the pre-master secret and figures out the session key. By that point, the server sends its *ChangeCipherSpec* message, thus, it is telling the client that the following messages will be encrypted. Finally, the server sends its *Finished* message encrypted with the session key it has just computed as well as the same MAC so as to verify the handshake's integrity.

After these steps the DTLS handshake is complete. Both the client and the server have a session key which they are going to use to carry out an encrypted and authenticated communication.

2.3 PUBLIC KEY INFRASTRUCTURE

Public Key Infrastructure (PKI) is a set of hardware, software, policies and procedures which create, manage, distribute, use and revoke digital certificates as well as manage public-key encryption. PKI uses digital certificates which verify the identity of website's servers, users or devices and ensures the integrity of the transaction. The most relevant elements of PKIs are described below: Digital Certificates and Certification Authority

2.3.1 Digital certificates.

Digital certificates are a kind of electronic identification of any peer that wants to establish a secure communication with another remote peer. Digital certificates are used in asymmetric cryptography, which is a cryptography system that uses pairs of keys. The pair consists of a public key, which can be accessed by anybody, and a private key, which is kept in private by the owner. Any of those keys can either be used for encrypting or decrypting and anything encrypted with one key can only be decrypted with other and vice versa. Furthermore, one of the most relevant information that certificates contain is the peer's public key, then certificates allow communication parties to share their public key in a way that can be authenticated. These certificates contain a kind of certification by a trusted source which ensures that the public key shown in the certificate belongs to the subject indicated in the certificate, in other words, that the party is who it is supposed to be. This trusted source is called certificate authority (CA) and will later be explained.

2.3.1.1 X.509v3 digital certificate structure

There are different certificate formats, however, nowadays used certificates follow the standard format of X.509. The latest version of this format is version 3. RFC 5280 [3] defines the structure of a X.509v3 digital certificate, which is shown in figure 5:

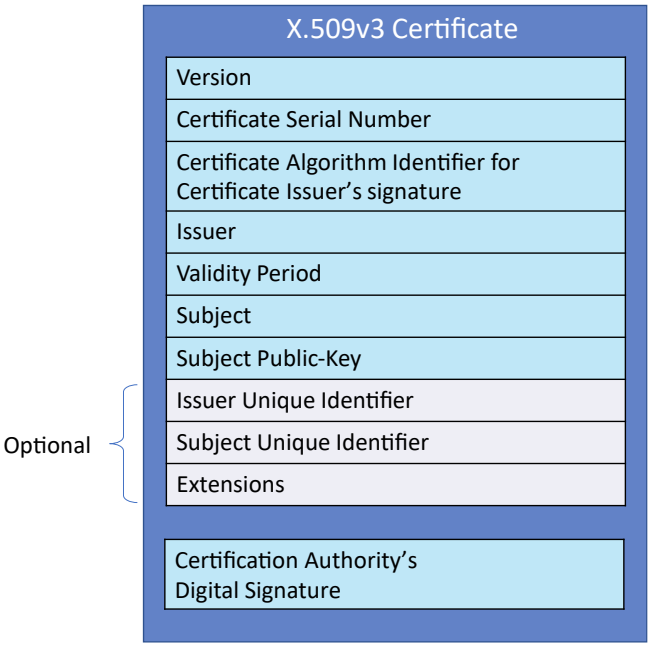

Figure 5: X.509v3 certificate structure

As figure 5 shows, a X.509v3 certificate has several fields, which are described below:

- **Version:** Certificate's version, actual one is 3.
- **Serial number:** A unique identifier for each certificate issued by a CA.
- **Algorithm identifier:** Identifier of the used algorithm to sign the certificate.
- **Issuer:** Distinguished name of the CA that has created the certificate.
- **Validity period:** It is represented providing the date and time when the certificate became valid and the date and time when it is no longer valid.
- **Subject:** The name of the website server, user or device that the CA issues the certificate to.
- **Subject Public key:** They public key that belongs to the subject's private key.
- **Certification Authority's Digital Signature:** Hash of the rest of the fields of the certificate encrypted with the CA's private key.

Any certificate modification can only and exclusively be performed by the CA that issued it. Moreover, any user with access to the CA's public key can retrieve the public key of the certified user in a reliable way. Once a communication peer (A) has obtained the other remote peer's certificate (B), it can prove the following:

- **Confidentiality**: As messages that A encrypts with B's public key are protected from eavesdropping by third parties.
- **Authenticity**: Messages that A obtains encrypted with B's private key have actually been issued by B.

2.3.1.2 Issues due to incorrect managing of certificates

In order to guarantee the security of the certificate-based authentication mechanisms, it is mandatory to be capable of managing certificates so as not to use an invalid one. Nevertheless, having a full vision into the number of certificate authenticating servers, users, devices and applications is a challenging task many organizations are not capable of carrying out. They do not often have such a visibility level or the ability to take instantaneous actions on certificates ensuring no significant risk of error. As Bill Holtz points out in this post [14], Certificate agility is just as important as crypto agility.

Not tracking and managing a company's certificates can lead to serious problems. Service interruptions due to certificate management errors are fairly common issues. For instance, Microsoft Teams went down after they forgot to renew a certificate [15]. Furthermore, California under-counted Covid-19 cases [16] due to the use of an expired certificate. Among many other errors, Spotify also presented an outage because of using an expired certificate [17]. They luckily managed to recover from the outages, but this scenario should be avoided altogether. Unfortunately, any organization with not much visibility on its certificates is exposed to the risk of suffering from outages. Moreover, some factors such as certificates' short lifetimes make certificates' management more difficult. Figure 6 shows the highest security concern about digital certificates.

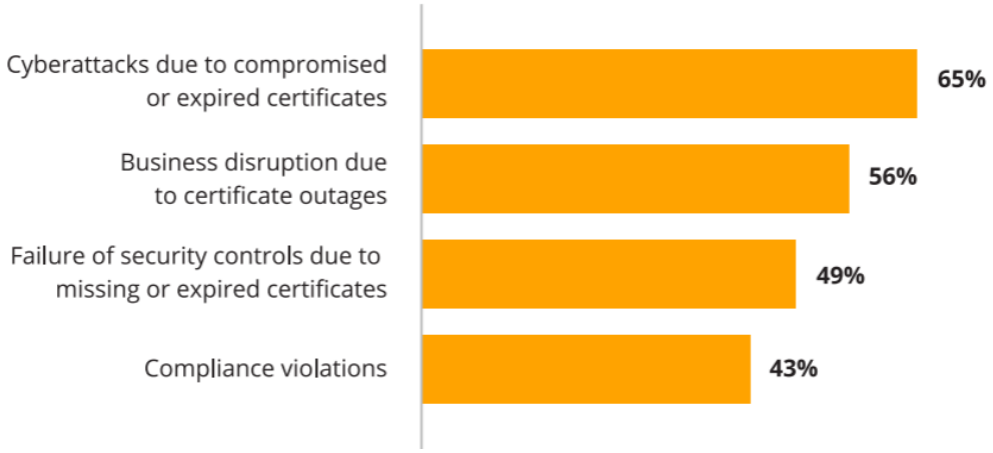

Figure 6: Top security concerns a survey participants voiced related to the widespread use of digital certificates [18]

2.3.2 Certificate Authority -- CA

CAs are significant entities in Public Key Infrastructure that issue digital certificates for subjects (servers or clients). A CA certifies that the public key included in a certificate belongs to the entity noted in it. Users can know CA's public key and so, they can verify if a received certificate is signed with the CA's private key. Browsers keep a list of well-known CAs' root certificates. As already mentioned, digital certificates present a validity period. A CA usually issues a new certificate before the old one expires. However, these certificates sometimes need to be revoked or replaced before their validity period expires due to several reasons. It could be the case that user's or CA's private key has been compromised or the user is no longer certified by the CA in question among other reasons. PKI standards define a revocation infrastructure so as

to provide security and to avoid using a certificate whose private key has been compromised or is no longer needed among others. CAs keep a list of revoked but not expired certificates issued by them. That said, whenever an entity receives a certificate from another party, apart from validating its signature, it must check if the provided certificate is not revoked. This can be performed downloading Certificate Revocation Lists [3] from a central CRLs Distribution Point or querying an Online Certificate Status Protocol service, whose use is more common than CRLs'.

In brief, a CRL is a blacklist of X.509 digital certificates that a CA revoked to let communications parties know that certificates on that list are no longer trustworthy. In order to check the status of a certificate using a CRL, the client contacts a CA or CRL issuer and downloads its certificate revocation list. After, it is the client who must search through the entire list if the certificate of the peer it would like to communicate with appears on that list. As it is a slow procedure, the client is allowed to download up-to-date CRLs only once per day.

Furthermore, CAs are helped by other two entities to perform their actions: Validation Authority (VA) and Registration Authority (RA). An VA is an entity responsible for providing information about the validation state of a certificate. It does neither issue nor revoke certificates, but it validates them. It carries out a real-time lookup of a certificate status in a database and answers back to OCSP requests with that information claiming if certificate is good, revoked or unknown.

Finally, another entity that supplements CA's function is the Registration Authority (RA). It is responsible for receiving certificate signing requests from the user as well as verifying those requests and forwards them to a CA. It also receives other certificate management functions such as revocation.

Both RA and VA are usually separated from the Certificate Authority for security and accessibility reasons.

2.3.2.1 How OCSP works

OCSP [19] is a protocol used by Certificate Authorities to check the revocation status of an X.509 digital certificate. It was created as an alternative to Certificate Revocation Lists, as it responds with more appropriate revocation information than the information provided with CRLs, it may also be used to obtain additional status information. A client sends a request about the status of a certificate to an OCSP responder. It accepts no certificate until obtaining the corresponding OCSP response. The protocol specifies how the transferred packets between the OCSP requester, and the responder look like.

Figure 7 shows how is OCSP implemented in PKI infrastructure. Considering that a client and a server are carrying out a DTLS or TLS handshake. At the point that the client receives the server's certificate, it wants to check its revocation status. Then, the client creates an OCSP request that contains the server's certificate serial number and sends it to the OCSP responder. The latter reads the certificate serial number from the request and checks its status with a trusted Certificate Authority. It looks for that number in a database usually managed in a CA. This database keeps records of all issued certificates by that CA. After that, an OCSP response is built, signed by the CA and sent back to the client. The response could be either *good*, *revoked* or

unknown. When the client receives this response, it verifies the CA's signature with the CA's public key. If the response status is *good*, they can go ahead with the handshake, otherwise, they can neither continue with it nor establish further encrypted communication.

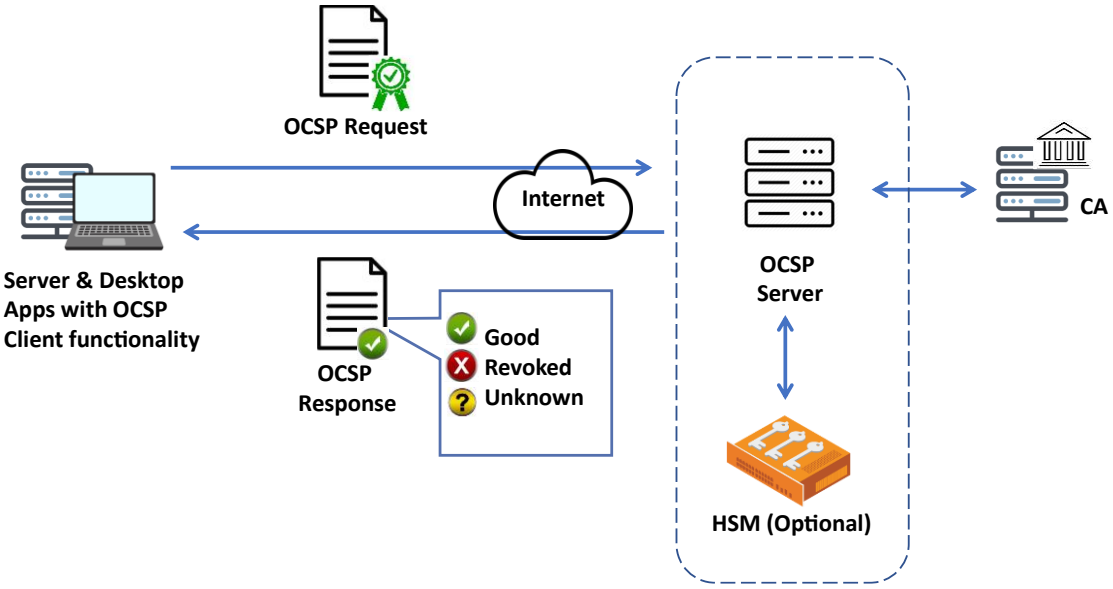

Figure 7: OCSP checking architecture

2.3.2.2 OCSP in IoT scenarios

The outbreak of the Internet of Things has resulted in more devices to manage, thus, more certificates to handle. Even though PKI certificates are considered as the best available technology to secure communications in IoT, its application in this scenario is not as simple as it seems, since IoT devices are neither as intelligent nor as powerful as a computer can be.

Revocation in IoT scenarios is likely to be performed using an OCSP responder rather than downloading CRL files since it is unfeasible for an IoT device to handle those significant lists. Even if OCSP is much more convenient than downloading and checking big CRL files, it can present some serious shortcomings:

- An OCSP responder may not be accessible through the IoT device's private network. Moreover, if it is connected to a server using a VPN all the possible OCSP servers' route must be provided.
- Another efficient way for the client is to use OCSP stapling. Instead of making the client check the revocation status of a certificate, the OCSP result is directly presented via the DTLS protocol by the DTLS server. The latter will send its status using a TLS extension, thus, no need to contact any external service. Nevertheless, only few DTLS implementations support OCSP stapling [20].

- Finally, using OCSP involves a significant computing cost, which makes it difficult for IoT devices to implement, as they do not have the same computational power as a computer, for example.

2.4 SOFTWARE DEFINED NETWORKING – SDN

In recent years new sources such as cloud, big data or IoT have entered the field of communication. Focusing on IoT, new challenges different from traditional Internet are posed such as application specific QoS requirements, diverse communication technologies, large amounts of data, and network conditions which are unpredictable. Therefore, traditional infrastructure becomes inflexible and difficult to adapt, consequently, network infrastructure demands changes in order to adapt to these new scenarios. The adjustment is not that simple, the fact that network infrastructure is made up of different equipment, usually from different vendors make it such a challenging task to carry out. Whenever both equipment and devices need to adapt to new requirements, someone needs to perform this new change or configuration in each device.

Bearing in mind all the issues and challenges IoT involves in traditional network infrastructures, Software Defined Networking (SDN) adapts remarkably to this scenario [21]. SDN consists of a network architectural method where data and control planes are separated by programmable switches placed between the planes in order to direct and modify data forwarding. They enable a flexible control of the network using high-level policies, without any regard for low-level configuration issues. In addition, SDN makes it considerably easier to manage network traffic in tricky networking topologies as there is a centralized panel instead which lets network administrators forget about handling each device manually. Thus, IT managers are able to adapt easily to demanding business and application needs such as bandwidth requirements. It is worth mentioning that the fundamental concept here is data flow.

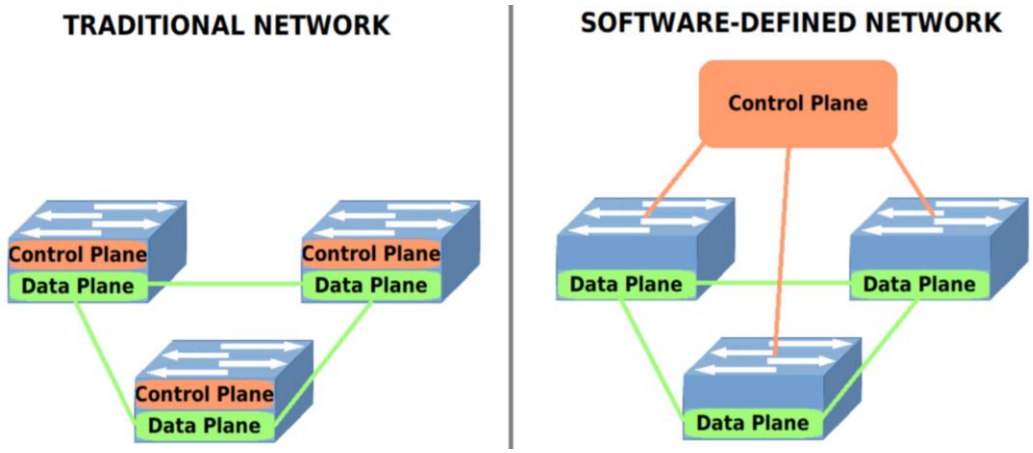

Figure 8: Comparison between traditional network architecture and SDN architecture

Furthermore, traditional networks were based on hardware, they consist of physical devices such as switches and routers, which have limitations when it comes to being handled. As figure 8 shows, data plane and control plane elements of any networking architecture were placed together, usually in the same code which was distributed by some vendors. Since SDN is software-based, all devices and equipment can be virtually configured and managed by a control plane, hence, they can be updated up to demanded requirements. In addition, SDN network devices do not need high-level algorithms to guess a packet destination as traditional routers do, it is the SDN controller which centrally manages packets on each device based on their configuration. What is more, an SDN controller can improve security in IoT scenarios since it provides considerable debugging tools.

2.4.1 Architecture

SDN allows the separation between the control and data, unlike traditional networks where both planes were deployed as a unique integrated system. This way, the controller provides more information about the status of the network. SDN architecture consists of three stacks. Figure 9 shows SDN layers.

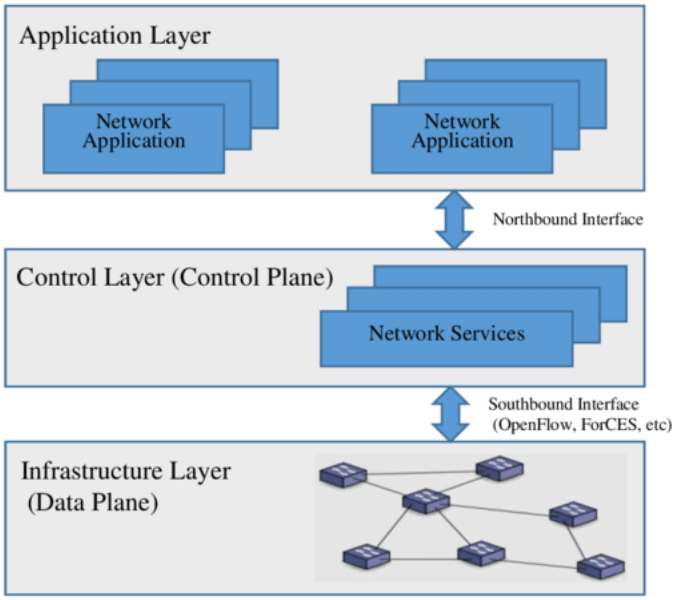

Figure 9: Software Defined Networking architecture

- **SDN Application Layer:** This layer consists of programs, better known as applications, that communicate with the SDN controller via APIs. The purpose of these applications is usually to manage the network, build analytics or handle business applications which run great data centres. It allows network managers to develop applications which provide further information for the controller. Furthermore, applications are capable of creating an abstracted view of the network. Thanks to the Northbound interface, applications can collect information from the controller to make decisions.

- **SDN Control Layer (Control Plane):** This layer hosts the SDN controller. This controller is a logical entity which a SDN application sends both instructions and requirements to it. Once it receives information from the application layer, it replicates that information to networking components. Not only does the controller communicate with the application layer, but it also does with the infrastructure layer (Data Plane). SDN controller retrieves information about the network from the hardware devices and sends it back to applications. Hence, applications can build an abstracted view of the network such as statistics and events about what is going on through the network.

- **Infrastructure layer (Data Plane):** The Data plane contains networking devices. They are responsible for performing the forwarding and data processing actions. These networking devices communicate with the SDN controller through the southbound interface. Since SDN is a virtual network overlay, there is no need to place all those devices in the same physical location.

Both the northbound and the southbound interfaces provide communication between SDN architecture's layers. On the one hand, the northbound interface is usually implemented by a REST API, which lets applications access the tools that the SDN controller exposes. On the other hand, the southbound API is likely to be implemented by OpenFlow. In addition, when there are more than one controller and they need to communicate with each other, two new interfaces are added: the eastbound and the westbound interfaces, which let this communication between controllers. Furthermore, all interfaces are open source and do not depend on the underlying hardware details.

2.4.2 OpenFlow

OpenFlow [22] is a communication protocol and open-source standard which enables data transfer between an SDN controller and the forwarding plane of network devices, both physical and virtual. It must be mentioned that OpenFlow is not the only protocol that composes SDN. It was created by researchers from Stanford University and its first specification was released in 2008. This protocol is standardised, marketed and promoted in production networks by the Open Networking Foundation (ONF).

The aim of these researchers was to have a set of devices only with a data plane which answer back to commands sent by a logically centralised controller implementing the single control plane of such a network. Thus, network capability can be dynamically increased. It is those commands and their responses that constitute the OpenFlow protocol. Running a centralized control plane turns out to be better than running a distributed control plane in every switch. It is much faster since a single point runs the algorithms and distributes the routes to every device in the network, in contrast to running distributed algorithms on every node and waiting for those to converge. In addition, OpenFlow is capable of completely replacing layer 2 and layer 3 functionality in commercial routers and switches.

OpenFlow goes over SSL, transferred data is encrypted, so as nobody can impersonate the controller. As it is shown in figure 10, there is a Flow Table in the switch, which would be like a forwarding table in a typical switch whose entries are designed as one wishes. The controller

tells the switch how it must behave, in other words, it is responsible for sending actions to populate and modify this Flow table. An OpenFlow switch can have more than one Flow Table, better known as a Flow Table Pipeline.

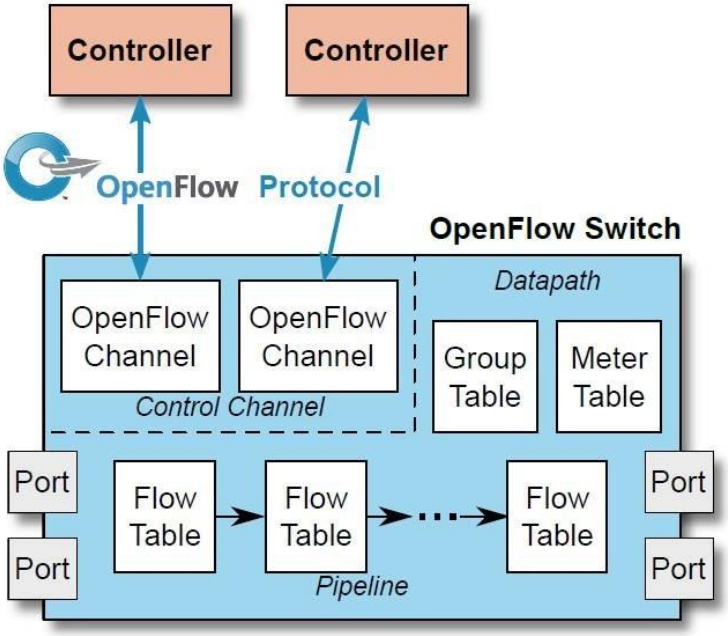

Figure 10: OpenFlow architecture

Flow Table entries are associated with at least one action. Whenever a datagram arrives, it is compared with all table entries, if it matches an entry, the corresponding action is carried out, otherwise, the packet is sent to the controller. It is worth noting that the packet is not analysed at network, transport or any other level, but all desired fields in the layer 2, 3 and 4 headers at once. Hence, one can define a rule and the desired fields to analyse according to his preferences (network preferences). The following actions can be applied to a packet:

- Forward:
  - **ALL**: Send the received packet through all switch's interfaces except the one it was received on.
  - **CONTROLLER**: Encapsulate the packet and send it to the controller. This enables packet_in/packet_out behaviour.If this action is performed, some extra headers must be added to the original packet defined as packet_in and packet_out headers respectively.
  - **LOCAL**: send the packet to the local network stack of the switch.
  - **TABLE**: execute actions on the flow table.
  - **IN PORT**: Send the packet through the same port it was received.
  - **OPTIONAL**: Typical forwarding, as a traditional switch or router.

- Discard: Whenever a flow entry specifies no action to perform, it means that all packets which match that entry must be dropped.
- Modify: Modify values of the packet header such as VLAN identifies, establish destination IP or priority.
- Queue: send the packet through an associated queue with a port

Switching

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP src	IP dst	IP Prot	TCP sport	TCP dport	Action
*	*	00:1f:.	*	*	*	*	*	*	*	port 6

Flow Switching

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP src	IP dst	IP Prot	TCP sport	TCP dport	Action
Port 3	00:20..	00:1f:.	0800	Vlan1	1.2.3.4	5.6.7.8	4	17264	80	port 6

Firewall

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP src	IP dst	IP Prot	TCP sport	TCP dport	Forward
*	*	*	*	*	*	*	*	*	22	drop

Figure 11: Examples of OpenFlow table entries

Figure 11 shows some common examples of Flow Table entries. It is worth mentioning that flow Table entries have a priority level so as to face the problem of overlapping entries. Moreover, they also have limited expiry time associated. Hence, the controller keeps track of information about the flow entry such as its active duration as well as statistics about the number of packets and bytes that have matched that flow entry.

2.4.3 SDN in IoT scenarios

Thanks to SDN structure, as already mentioned, the controller allows splitting a network into distant subnets, a separation that IoT demands somehow. This controller can communicate with the IoT application over the "Northbound API". All in all, SDN plays a major role in IoT due to the remote control of network setup with no need of contacting directly with IoT devices. Figure 12 shows how SDN is integrated in an IoT scenario.

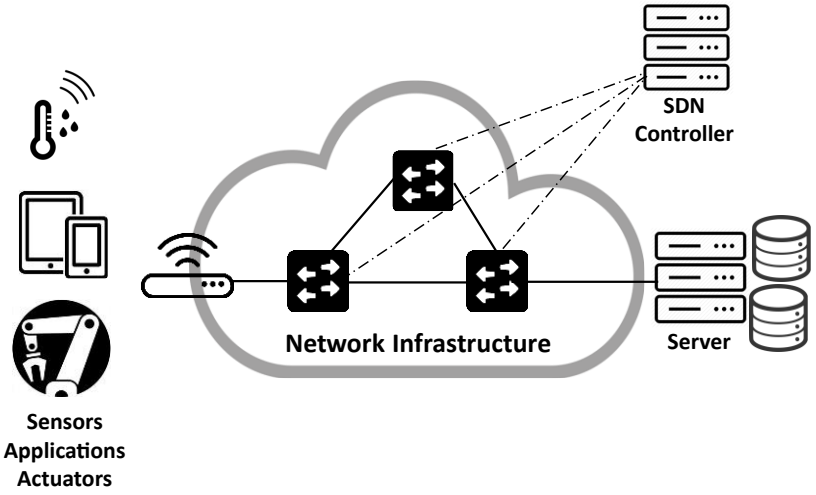

Figure 12: IoT integration in SDN

2.5 IN-NETWORK COMPUTING – INC

As previously mentioned, OpenFlow is a fundamental pillar in Software Defined Networking. Nevertheless, it presents some issues. Openflow assumes that switches have a fixed-well-known behaviour described in their relevant datasheet. They implement fixed IEEE IETF standard protocols in silicon. There is no possibility of changing their behavior and adding new protocols or control datapath. It allows the controller to add and delete forwarding entries for about 50 types of headers. In order to support new networking protocols, one must move to an OpenFlow new version. Furthermore, there is limited interoperability between vendors as there are differences in the southbound interface handled at the controller, thus, this brings them complexity. Therefore, there was a need to make the data plane programmable as well, which makes a new technology come along: In-Network Computing (INC).

In-Network computing, also known as In-Network computation or Netcompute, is an emerging researching area that consists of offloading standard applications to run within network devices. Thanks to INC, user data is terminated before it reaches the host, thus, it saves CPU cycles and frees up resources. Until today, INC has been implemented in FPGAs, SmartNICs and switch-ASICs. Below, data plane programming will be explained as it is the core of INC.

2.5.1 Data Plane programming

Both SmartNIC and programmable switch-ASIC have been the architects of network computing. They are programmable network devices that allow network administrators to implement functionalities on them up to their whim writing code using high level languages. The administrator programs its data plane using a new coding language created for this purpose, P4, which will later be explained.

2.5.1.1 Benefits of Data Plane Programmability

Data Plane programmability presents several advantages, the most relevant ones are listed below:

- New protocols can be added in the future with no need of updating anything.
- Network complexity can be reduced as unused protocols or features can be removed.
- Resources are efficiently used thanks to table flexibility. For instance, they can be scaled up for v4 or v6 IP addresses.
- A greater visibility of the network is provided as statistics are obtained much more efficiently.
- Development of software style, it is possible to update the Operating System running on a switch.
- There is no need to share one's API, then, one keeps its own ideas.

One must think in programming rather than protocols.

2.5.1.2 P4 language

As already mentioned, a new language to program network devices' data plane is needed, thus, P4 [23] was created. P4 stands for Programming Protocol-independent Packet Processors, which is a domain-specific language for network devices that specifies how data plane devices such as switches or routers, among others, process packets. In a nutshell, it allows the control plane and the network operators to decide exactly what the networking devices are supposed to do. P4 was not created to replace OpenFlow, but rather to solve its issues and provide extra functionalities to SDN. It addresses a different need: programming the data plane. This language does not assume any specific protocol, it is up to the developer to design which headers to program and which networking devices are going to support them. Thanks to P4, apart from being capable of implementing specific behaviour in the network, changes on it can be carried out quickly, in the order of minutes.

A fixed-function switch was based on a bottom-up design, the switch datasheet makes the rules, whereas a P4 programmable switch follows a top-down design. In the latter, the user or controller decides how the network processes packets. Figure 13 represents interactions between management, control and data plane in data plane programming. It shows that SDN benefits are kept such Control and data plane splitting or control centralization among others. In addition, a P4 program must be first coded and then it is sent to a P4 compiler. This is responsible for translating it to some representation so the data and control plane can understand each other. Unlike with a fixed-function device, one has control over which features the network supports.

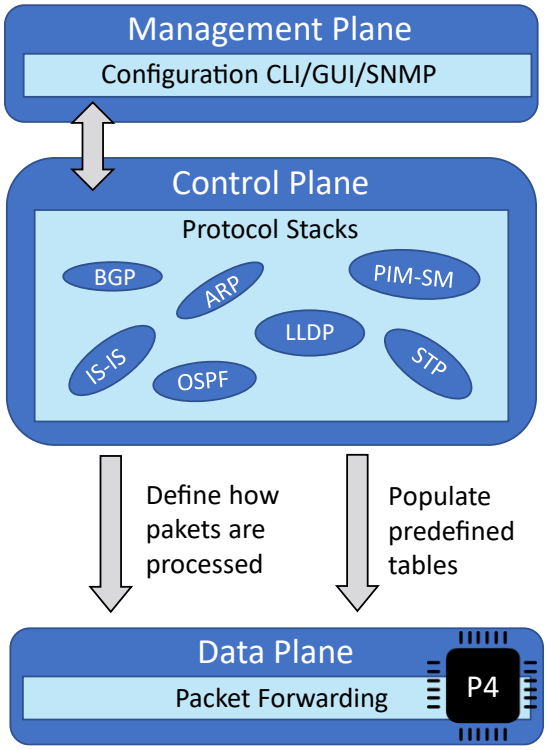

Figure 13: Data plane programming

2.5.1.3 P4 targets and architecture

On the one hand, P4 does not depend on low-level placement details, it is target independent. No matter which programmable hardware is used to implement the data plane, as long as it has a programmable data plane. A typical example of implementing a switch target is using Behavioural Model version 2, BMv2.

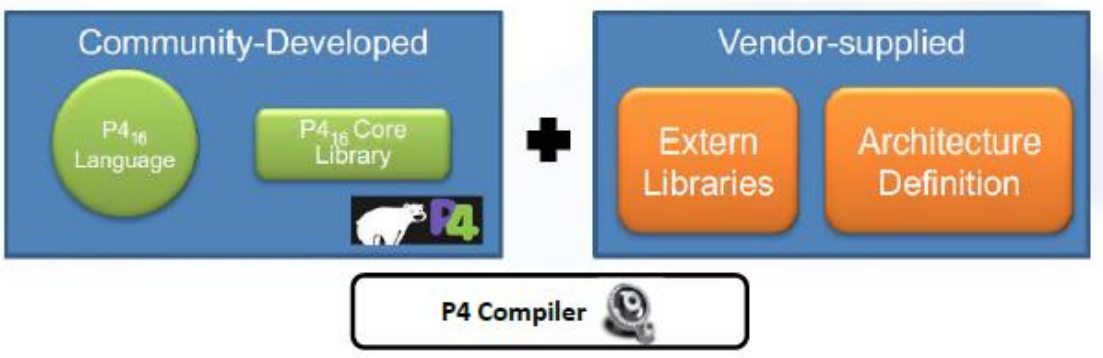

Figure 14: Target-independent vs. Architecture-dependent

On the other hand, P4 does depend on fundamental metadata and externs, it is architecture dependent. By extern, it means external parameters passed to the switch instantiation. Some extern types are checksums, hashes or counters among others. An architecture is composed of a set of P4 programmable components and their interfaces for the P4 programmer. Figure 14

shows the elements of the programmable switch that belongs to the target and the ones which are part of the architecture supplied by the vendor. For instance, V1model represents a P4 architecture. The elements that compose V1model architecture are shown in figure 15.

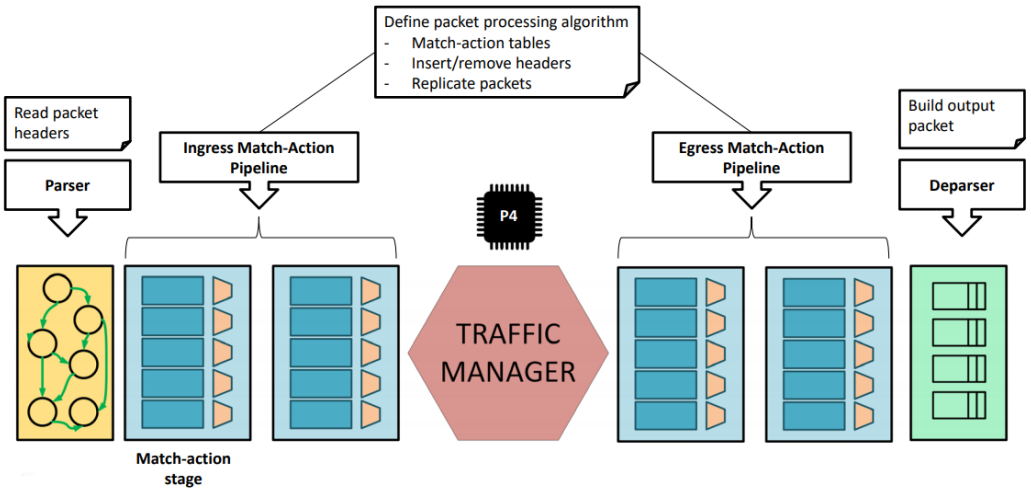

Figure 15: P4 V1model architecture

As V1model architecture is used in this project, its elements and their functionalities are described below:

- **Parser:** Unlike OpenFlow, there are no assumptions made about the kind of packets the switch is going to be processing in the network, it is up to the P4 programmer to define a programmable parser. A parser is like a state machine in which the programmer specifies which packet format the switch accepts in its packet processing pipeline. Typically, there is one state for each different protocol. In each state, the switch extracts information from the packet and then makes a transition decision based on that information.
  A good example to understand it could be a program which is only going to process ethernet packets. The entry point on the parser will be a state which will extract 14 bytes from the packet (6 of source MAC, 6 of destination MAC and 2 of Ethertype). The parser is going to look at the value of ethType, then, based on the value of this field, it is going to extract IPv4, IPv6 or any custom header.

- **Ingress/Egress Match-Action Pipeline:** These programmable match-action pipelines are like a reuse of a Match-Action abstraction from Openflow. They implement packet processing algorithms. In order to implement them, they take the extracted headers and run them to a sequence of Match-Action tables which are going to modify the headers. Usually performed actions are inserting, removing and modifying headers or clone packets among others. It is worth noting that from one Match-Action to the other Metadata can be retained. Metadata consists of some extra fields included in the packet which are useful to determine what to do next with the packet.

- **Deparser**: The deparser takes the previously modified headers and turns those headers back into a packet stream and appends the payload then.

In brief, firstly, the received packet is parsed into individual headers according to the parser's state machine. Next, those headers can go through multiple Match-Action tables, which modify them. Finally, the packet is serialized by a programmable deparser.

2.5.1.4 P4 Runtime

OpenFlow acts as an API which makes it possible for the controller to communicate with the data plane, in other words, OpenFlow populates a switch's fixed tables. Nevertheless, in order to be able to both populate predefined tables and define how packets are processed in a programmable switch OpenFlow presents some issues. It is protocol-dependent, protocol headers and actions are included in the specification. Hence, another model must be used: P4 runtime. It is a framework for runtime control of P4 targets which consists of an Open-source API and a server implementation. The API does not change with the P4 program, it is the same API for all P4 programs. Thanks to it, a new P4 program can be pushed without recompiling the software stack of target switches. In a nutshell, P4 runtime service allows a local or remote entity to load a program, send or receive packets and read and write forwarding table entries, counters and other elements. Figure 16 shows in a schematic way the elements that take part it the process of programming a programmable switch with P4 language.

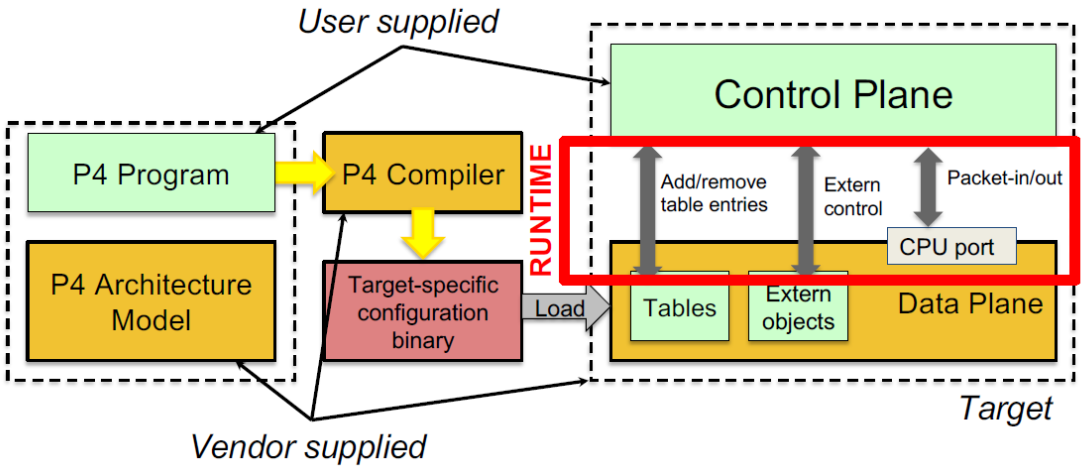

Figure 16: P4 target

3 OBJECTIVES

3.1 MAIN OBJECTIVE

The purpose of this master's thesis is to design and implement a digital certificate validation system for IoT devices delegating client's certificate validation actions to a network element such as a programmable switch. Since certificate validation actions are costly operations for IoT devices, it is more suitable that a more powerful device does them. In addition, this action delegation is implemented in a transparent way to communicating endpoints and does not affect communication security. In order to achieve this goal, various partial objectives have been defined so as to develop the project in a clear and organized way. Once a partial objective is met, one can move to the following.

3.2 SECONDARY OBJECTIVES

For a structured performance of the project, the following stages have been established with an objective per stage. Each of them will contribute to the achievement of the desired main goal.

3.2.1 Architecture design

Firstly, the architecture of the network must be designed. As already mentioned, this project uses Software Defined Networking and In-Network Computing. Therefore, a deep research on those technologies must be first carried out so as to know the features they offer as well as their benefits and limitations in a network design. Furthermore, it is also necessary to propose a proper design in terms of network performance and security. For instance, it is important to take into account the number of hopes each packet sent through a network undergoes to reach the destination peer or if network devices and communications can be subject to threats.

3.2.2 Study of the available tools and selection of the best alternatives for later implementation

Once the architecture design is clear, the second secondary objective is to perform a study of the available technologies and tools for its implementation. On the one side, technologies for both SDN and INC implementations must be analysed as well as their hardware requirements such as CPU or RAM memory and programming languages. On the other side, a study of available software for the services that the network will offer must be carried out. The main applications/services the designed network must provide require software tools to execute a DTLS handshake and software that offers certificates' validation service.

3.2.3 Network model implementation

After having selected both the technologies and software tools to use, the designed network model can be implemented. In order to do it in an organized way, the following steps are followed during the implementation of the solution. First, a simple SDN network must be deployed just letting a client and a server execute a handshake with a programmable switch placed between them which is programmed to forward packets. Secondly, the switch data plane will be programmed somehow that it identifies the received certificate form the server and sends it to the controller. After that, an OCSP responder is introduced to the simple network architecture. Therefore, some functions are implemented at the controller so as to validate received certificate and check its revocation status querying the OCSP responder. Eventually, the handshake will be executed in the final scenario.

3.2.4 Validation of the implemented solution

Once having checked that everything works, some tests will be carried out so as to have enough data about the time it takes to perform a DTLS handshake and validate certificates in the designed architecture model scenario. In addition, another scenario will be implemented, that consists of executing on the client side validation actions carried out by the controller. Same performed tests in the first scenario will be carried out on the second scenario. Therefore, information about DTLS handshake and certificate validation timings will be obtained in each scenario for its later evaluation.

3.2.5 Analysis of results and conclusions drawing

In the light of the obtained results, the implemented system assessment can be made. It will be made a comparison of timings obtained between the previously mentioned scenarios. Based on those results and evaluations, project's conclusions will be drawn.

4 BENEFITS

In this section this project's benefits are described. They have been classified in three categories, technical, social, and economic benefits.

4.1 TECHNICAL BENEFITS

This project brings several improvements to IoT communications as it provides an efficient and secure integration of IoT devices in the Internet. The following ones are considered as technical benefits since they have impact on used technologies.

1. Revocation methods of Public Key Infrastructure do not currently work well for IoT scenarios due to the lack of ability in those devices to carry out some heavy operations such as checking certificates' revocation status together with the signature validation. Nevertheless, as this project deals with the implementation of OCSP revocation method in IoT scenarios, it represents an innovation in terms of Public Key Infrastructure. Not only is it an alternative to IoT devices, but it also provides a broad alternative to any device which is not able to query a revocation system and understand its answer or any scenario where it becomes hard for the client to check certificates signature.

2. As a result of the previous benefit, with regard to security, IoT security will certainly be improved. Outages in operation will be mitigated as it will be more challenging to attack those devices. Thanks to having access to a revocation system, it is rather unlikely to use a revoked certificate whose private key has been compromised and therefore suffer from a Man in the Middle attack. Furthermore, it will also avoid the use of fake certificates which are not signed by a trusted CA as the validation of certificate's signature will betray them.

3. Thanks to using a programmable switch, any changes in the scenario can be easily carried out since the switch can be programmed up to the programmer's needs. Then, it will quickly adapt to the new scenario.

4. This solution is focused on delegating certificates validation actions to a networking element such as a programmable switch. This delegation can be adapted to any other scenario different from certificate's revocation checking. Anytime a client, host or whatever connected element to a network is not capable of performing an action due to its lack of intelligence or due to its high traffic load, among other reasons, the programmable switch can act as a load balancer or "action balancer" in not very complex networks. Otherwise, delegating many client's actions to a switch will overload it.

4.2 ECONOMIC BENEFITS

Whenever a solution to a problem is proposed, the economic aspect is always considered. If it is all about economic losses, it is rather difficult to implement the solution. Hence, this project also presents the following economic benefits:

1. One of the most relevant economic benefits is that there is no need to buy extra equipment as SDN is based on software and using a programmable switch does not involve any cost. Then, it brings technical benefits with no or little financial cost.

2. As already mentioned in section *1 Introduction*, the use of a revoked certificate may lead to a cyber-attack such as a man in the middle attack. It is also known as identity theft, phishing. A client believes it is communicating with a trustworthy server instead of an adversary and starts sending confidential data to the attacker since IoT devices collect sensitive user or environment information. A cyber-attack usually results in significant economic losses, e.g., the production of a factory may have to be shut down or confidential data may be leaked and the company's image damaged. The less likely it is that an IoT device will suffer from an attack, the better off the company's economic situation will be.

3. Hand in hand with the previous benefit, the following is presented. A compromised IoT device due to certificate management errors can lead to service outages. As soon as the owner of an IoT device realises that it is compromised, he may decide to get rid of it and replace it by a new one, which in turn, involves a cost. Therefore, preventing such attacks will in turn prevent the purchase of new devices, which means savings either for a company or for a user.

4.3 SOCIAL BENEFITS

As far as social benefits are concerned, several benefits have been identified as well.

1. The number of IoT connected devices is considerably growing. More and more people and companies are making use of this technology. Therefore, security in IoT devices is becoming a concern in society. This project brings more security to IoT devices, which at the same time, reduces the collection of sensitive data by third parties. Users will benefit from IoT services in a secure way, which will somehow encourage its use.

2. IoT devices are often used to collect personal data such as wearable that measures heart rate, temperature, sleep, etc. In order to see this information on a mobile application, the device has to transmit it and therein lies the risk that this communication is not secure. Not securing this communication can lead to data leaks, in other words, failure to provide data privacy of data. This project improves user's data privacy, which is significant to take into account so as to enforce personal data protection law, which guarantees and protects the processing of personal data and the fundamental rights of

people, in particular the right to honour and to both personal and family privacy. Therefore, ensuring that the solution of this project stays away from breaching data protection law generates a great deal of peace of mind for users.

3. Apart from being a good and secure service for the user, it also brings great benefits in terms of complaints, prosecutions, and convictions. The more barriers an attacker must face, the less likely it will be able to commit the attack. This will reduce many of the complaints owing to attacks on any IoT device. It will minimize prosecutions due to accessing or modifying important company data as well. All of this, will decrease cybercrime targeting IoT devices.

5 REQUIREMENTS

This project, in accordance with the described objectives in section 3, must meet certain requirements which somehow condition the design and implementation of the system. This said, the validation system must meet the following requirements:

- Maintain communication standards. One can neither modify nor delete fixed exchanged messages that are established by protocol. It is not either allowed to add new messages. For instance, connection establishment messages or exchanged messages between two parties during its handshake.

- Any software used in the model must be Linux compatible. This requirement has to do with the used software for network emulation, described in section *6 Alternatives analysis*. The network emulation software lets run most of the installed software on the host machine, the only drawback is that programs must be able to work in Linux Operating System.

6 STATE OF THE ART

This section provides an overview of research papers of similar projects where SDN and/or INC are used to offload IoT devices from costly computational operations.

Lightweight Edge Authentication for SDN [24] proposes a solution which delegates to a P4 programmable switch security actions that usually require sophisticated equipment. The chosen security action is port knocking, which is typically used to protect services that must be exposed to the internet. It is a mechanism that allows opening a firewall port by following a certain series of attempts to connect to ports that are closed. Only authorised persons will know on which port sequence connection attempts must be made to finally enable the firewall. In this case, port knocking has been generalized to behave like an authentication method for hosts or subnetworks that try to establish a connection to a specific server. A unique SDN switch is programmed using P4 in order to perform the port knocking application an make it behave like an authentication unit on the network ingress. This is not the only solution that implements port-knocking actions in a programmable switch, other researchers proposed *P4Knocking* [25], whose aim is to offload host-based firewall functionalities to the network. Therefore, firewall functions are deployed on the device data plane, relieving hosts from coping with undesired traffic. The way this solution deploys the port knocking service is more transparent and efficient than a port knocking implementation based on host. Moreover, it provides a high flexibility and portability with local or remote control planes since it demands no specific purpose externs apart from registers. Finally, so as to ensure its benefits, this paper presents four different implementations involving the data and control planes to varying degrees.

Implementing firewall actions in a programmable switch turns out to be very useful that more proposals address, such as *P4Guard* [26]. *P4Guard* is a virtual configurable firewall based on Network Function Virtualization with Software Defined Networking. It is a solution to several problems that virtual firewalls need to face. For instance, the demand of an efficient management of computer virtualization resources. This firewall is software-based and it specifies packet processing logic using P4 programming language. Thus, packet forwarding functions are developed in programmable switches' data plane using P4. Packet headers, parsers and processing behaviours of the proposed software firewall are defined there. Therefore, *P4Guard* can be used with any P4 compatible programmable switch regardless of the used protocol. This firewall has a controller that starts and removes dynamically firewalls in a network. Moreover, it can also be easily configured to update its functionalities and install new dynamic firewall rules on the go.

Another way of providing security using programmable data planes is classifying and sampling traffic. There is an implementation of a SDN packet forwarding Verification Mechanism [27]. This verification mechanism is carried out at the device's data plane and consists of adding a cipher identification to the packet. The programmable P4 device is added to a SDN network, and it samples network traffic flow without affecting its typical forwarding. These packets are sent to the controller, whose task is to verify the integrity of the sampled packet and to send flow rules

to the OpenFlow forwarding device. These rules are provided to the implemented mechanisms at the switch's data plane. Thanks to both the rules and mechanisms, the programmable device can control suspicious data flows such as malicious tampering.
In comparison with resembling verification mechanism, presenting the same security verification processing overhead, a new proposal is addressed by other researchers [28]. It proposes a more fine-grained solution on the data plane as well. It consists of a low footprint and low latency traffic inspection mechanism for real-time DDoS attack detection. Both mentioned mechanisms for traffic inspection and sampling are implemented on the switch. However, a firewall is not the only way of implementing security, security tunnels are also a practical use of protecting from malicious tampering.

For instance, *P4-IPsec* [29] is a site to site and host to site VPN with IPsec in P4-Based SDN. It is a solution to some projects that investigate how to leverage the centralized control plane of SDN to simplify IPsec operation. IPsec deployment on SDN is limited due to the fact that SDN switches have a fixed function data plane that does not offer IPsec. Taken all of this into account, *P4-IPsec* proposes the implementation of IPsec in SDN using P4 programmable data planes. The solution provides ESP implementation in tunnel mode, and it supports several cipher suites. P4 switches are programmed somehow, they behave as IPsec tunnel endpoints. *P4-IPsec* uses a SDN controller that carries out required functions to configure and renew tunnel endpoints. One of the main objectives of *P4-IPsec* is to investigate how well it can be implemented on existing P4 switches. In this case, a BMv2 P4 target is used to implement the solution. A similar solution proposed by other researchers is *P4Sec* [30], a new mechanism of security tunnel forwarding on SDN using P4 programmable devices. Since P4 allows to program a data plane to perform packet forwarding actions, this solution creates a SDN security tunnel so as to avoid malicious data tampering or stealing, among others. For that purpose, a new protocol header, *P4Sec,* is defined and a P4-based SDN tunnel is created. Therefore, the packet routing and forwarding is based on the gateway's identity, which adds the *P4Sec* header. The device identity of the gateway is combined with the Diffie-Hellman (DH) key exchange algorithm to negotiate a tunnel session key so as to provide encryption. This solution is implemented on a BMv2 programmable switch, and it shows that *P4Sec* security mechanism guarantees authenticity, integrity and confidentiality.

To sum up, so far, P4 has been used to offload many different security operations to end devices. Nevertheless, there has not been found any where P4 is used for certificate verification in the DTLS handshake. Therefore, taking this into consideration, this master's thesis would be the first proposal of offloading security operations regarding certificate verification in DTLS handshake to a programmable switch using P4.

7 ALTERNATIVES ANALYSIS

This section presents an analysis of different alternatives that have been considered to design and implement the proposed solution. In this project, four main issues have been identified for later analysis: The required software to run a programmable switch inside an emulator of SDN, the software used to implement a DTLS handshake, the libraries for validating the certificate and finally, the necessary software to implement an OCSP responder.

7.1 PROGRAMMABLE SWITCH SOFTWARE

In order to implement a programmable switch in an SDN, the best option is to use a Virtual Machine provided by a P4 repository since they have all the necessary software already installed. To do so, two VMs have been analysed so as to implement a programmable switch. As both VMs use Mininet as SDN emulator, first Mininet is explained, after that, the analysis of two VM will be described.

Mininet

Mininet is a network emulation orchestration system. It provides a virtual test bed and development environment for SDN. It is capable of running several end-hosts, switches, routers and links on a unique Linux kernel. In order to make a single system take after a complete network running the same kernel, system and user code, Mininet uses lightweight virtualization. A Mininet host acts as a real machine. One can run programs which send packets through an interface that seems like a real Ethernet interface. Those packets are processed by a switch, router or middlebox that seems like a real one. In a nutshell, the behaviour of Mininet's virtual hosts, switches, links and controllers is similar to discrete hardware elements. They are software-based instead of hardware-based as figure 17 shows.

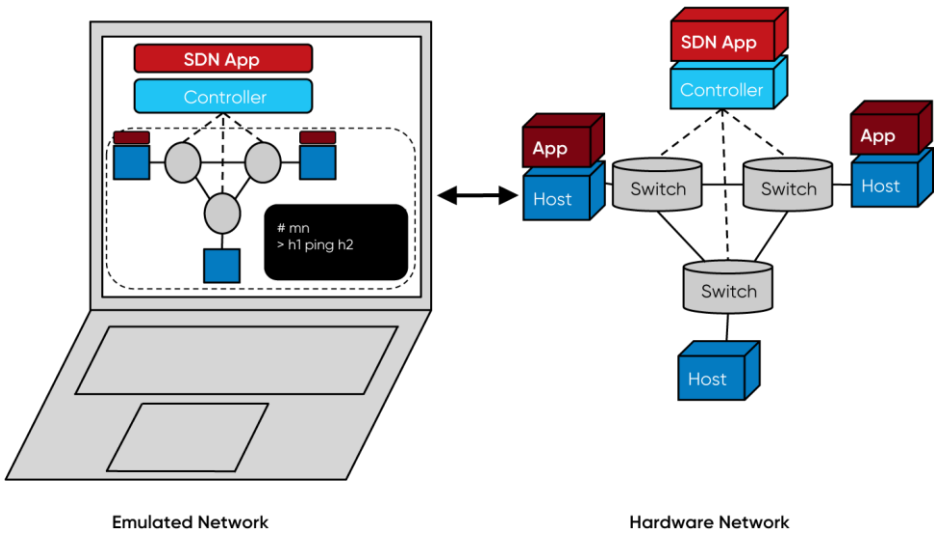

Figure 17: Mininet vs. traditional network

Mininet's most relevant features are described below:

- It is easy to create custom topologies and test them with no need to wire up a physical network, which is a good solution for complex topologies.
- It is possible run most of the real programs that run on Linux
- Since Mininet's switches are programmable, packet forwarding can be customized
- Mininet can be run on a computer, on a server, in a Virtual Machine, on a native Linux box or in the Cloud such as Amazon EC2.
- Mininet is an Open-source project. One can for instance fix bugs, add additional information.
- SDN designs can be moved from Mininet to the real hardware in live deployments
- Changing network configuration and running again the network is a quick task as Mininet takes only a few seconds to start up a network.

Eventually, Mininet has an extensible Python API used to create networks and release them under an authorizing Open-Source license. After having understood Mininet, *P4lang tutorials VM* and *P4-utils* are analysed below.

7.1.1 P4-lang tutorials VM

This Virtual Machine is provided by p4lang/tutorials [32] Github repository as a vagrant file. It provides all of the required software to implement a programmable switch in an SDN environment as well as to run Mininet SDN emulator.

7.1.1.1 Software / Hardware requirements

In order to install this VM, first a software designed to run and create virtual machines on a physical machine must be installed such as VirtualBox or VMware among others. With regard to hardware, the host machine where it will be installed needs at least 25 GB of free hard disk space, moreover, p4-lang VM is already created with 2 GB of RAM and it is recommended to give at least 2 CPUs to it. It is worth mentioning that installation process takes time to be completed.

7.1.1.2 Features

This VM provides several exercises which are useful to familiarize with P4 language. It presents the following benefits:

- Provided exercises present different application scenarios; thus, one can base its project in those, just modifying network configuration files and coding P4 program up to its whim.
- It presents a command-line launcher to instantiate networks.
- Networks are defined using json files.
- Allows to perform any configuration or installation action as one can access as root.

- Python 2.7 installed by default; however, any newer version can be installed.

Despite having considerable advantages, it presents a slight inconvenience: the incompatibility with other exercises developed with other P4 VMs such as *p4-utils*. The latter VM, which will be explained below, is based on created modules, which lets the developer implement many functionalities. However, exercises developed in that VM are not compatible with *p4-lang tutorial* since it should be reconfigured so as to be capable to act with modules such as *p4-utils*.

7.1.2 P4-utils VM

This Virtual Machine is provided by p4-utils [31] Github repository as a vagrant file as well. It has the required software to implement a programmable switch in an SDN environment as well as Mininet SDN emulator.

7.1.2.1 Software / Hardware requirements

In order to install this VM, as already mentioned in the former alternative, first a software designed to run and create virtual machines on a physical machine must be installed. As far has hardware requirements are considered, the VM needs 4GB of RAM and at least 2 CPUs. Its installation takes little time to complete, far less than the other VM.

7.1.2.2 Features

P4-utils creates virtual networks using Mininet and extensive nodes where p4-enabled switches can be run. Those switches are software-based such as Open vSwitch, Linux Bridge, or BMV2 switches. P4 utils was created with the purpose of making P4 networks easier to build, run and debug. It is based on P4 lang repository making much simpler to emulate a SDN with a programmable switch. P4-utils presents the following advantages:

- It presents a command-line launcher to instantiate networks.
- Host and Switch nodes are based on P4lang [32] repository.
- Networks are defined using json files.
- It improves Mininet command-line interface allowing to reboot switches with updated p4 programs and configurations, without the need to reboot the entire network.
- Both topology and network features can be saved in an object which can be loaded and queried to get significant information.
- In order to make P4 more modular, different modules are used, e.g, a module for the topology, another for the controller, etc. One can also create its own block.
- Moreover, some useful python methods are documented such as getting host's IP or the shortest path between nodes among others.

At the same time, it presents a major drawback which, in turn, triggers other drawbacks. The owner did not allow to execute some scripts or allow to install some software such as newer Python version. As this VM has Python 2.7 installed for Mininet deployment as well as for

controller functions, any library or used software must be compatible with Python 2.7. In addition, Python 2.7 is deprecated since January 2020, thus, it is not currently maintained, which means that it is strongly necessary that any software used is compatible with that python version. Unfortunately, that is not this project case, it uses many software tools which are implemented using Python3. Moreover, it presents the same issue as *p4-lang tutorials*, it cannot execute exercises developed in that VM.

Selection Criterion

Eventually, in order to make a comparison between these VMs, the following selection criterion have been applied:

- **RAM and CPU:** It is important that VM hardware requirements are satisfied by the computer one has, otherwise, another computer must be used. In addition, the more RAM and CPUs a VM have, the better performance it will present when running all the project scenario. Furthermore, it is always good to have extra RAM and CPU.
- **Installation process**: It refers to the time and complexity it takes to install the VM. This is a fact to take into account in case it is required to install it again at some point of the project. It could be due to the need of taking the way back to the starting point.
- **Ease of development and implementation:** This criterion involves the tools each VM machine provides to develop and implement a network. In other words, the less actions a developer takes to both develop and implement the system, the more hours of the project are saved.
- **Availability to install new software:** Let the developer install extra software so as to provide more functionalities than the ones Mininet, P4 and each VM provides by default.

	P4-utils	P4-lang tutorials
RAM and CPU (5%)	8	6
Installation process (5%)	7	5
Ease of development and implementation (10%)	8	7
Availability to install new software (80%)	2	10
TOTAL	3.15	**9.25**

Table 1: Evaluation of programable switch software

As table 1 shows, P4-lang tutorials has been chosen due to the high availability to install new software. A VM which presents problems when installing some new software that the project requires will end in failure in case there is no other alternative. Furthermore, p4-tutorials provides examples that can be easily modified and adapted to project scenario.

7.2 IMPLEMENTATION OF DTLS HANDSHAKE

This second alternative analysis that has been carried out concerns libraries that make it possible to implement a DTLS handshake. There are available repositories of developed programs which implement DTLS handshake, therefore, each repository will be analysed in terms of the programming language used, its libraries and compatibility with other software required in this project.

7.2.1 Scandium -- Californium

Scandium repository [34] is a subproject of Eclipse Californium project. Californium provides a framework to build an IoT application as well as examples on its Github repository, thus, one can contribute to them. It is licensed by Eclipse Public License and Eclipse Distribution License. Scandium implements DTLS 1.0 and DTLS 1.2 to secure an application with pre-shared keys, certificates or raw public keys. It has a Java socket abstraction for sending and receiving byte arrays in UDP, DTLS, TCP, etc. Then, it can be used to secure UDP to any type of application which goes over it.

7.2.1.1 Software requirements

As it is programmed using Java, it requires some extra software: Eclipse, Egit, m2e and maven so as to run Java programs. Required libraries to execute the DTLS handshake are defined in californium core project. Therefore, just cloning Californium repository [34] and building it will allow to use required libraries. It is worth mentioning that in order to build the repository java jdk11 or 15 is required.

7.2.1.2 Implementation

The DTLS handshake is performed running some java jars. Not only does it implement a handshake, but it also allows to send application data.

Figure 18 shows a DTLS handshake executed with californium repository provided jars.

Source	Destination	Protocol	Length	Info
192.168.56.111	192.168.56.110	DTLSv1.2	194	Client Hello
192.168.56.110	192.168.56.111	DTLSv1.2	102	Hello Verify Request
192.168.56.111	192.168.56.110	DTLSv1.2	226	Client Hello
192.168.56.110	192.168.56.111	DTLSv1.2	516	Server Hello, Certificate, Server Key Exchange, Certificate Request, Server Hello Done
192.168.56.111	192.168.56.110	DTLS	434	Certificate, Client Key Exchange, Certificate Verify, Change Cipher Spec, Continuation Data
192.168.56.110	192.168.56.111	DTLSv1.2	117	Change Cipher Spec, Encrypted Handshake Message
192.168.56.111	192.168.56.110	DTLS	186	Continuation Data
192.168.56.110	192.168.56.111	DTLSv1.2	82	Application Data

Figure 18: Exchanged message during DTLS handshake using Scandium repository

In this case, the server also requests the client's certificate in the 4th message and so, it sends it back in its response. This action is not required in this project. Paying attention at the DTLS message that contains server's certificate, it is only sent the server's certificate, as figure 19 shows. The client, as well, only sends back its certificate, no more certificates.

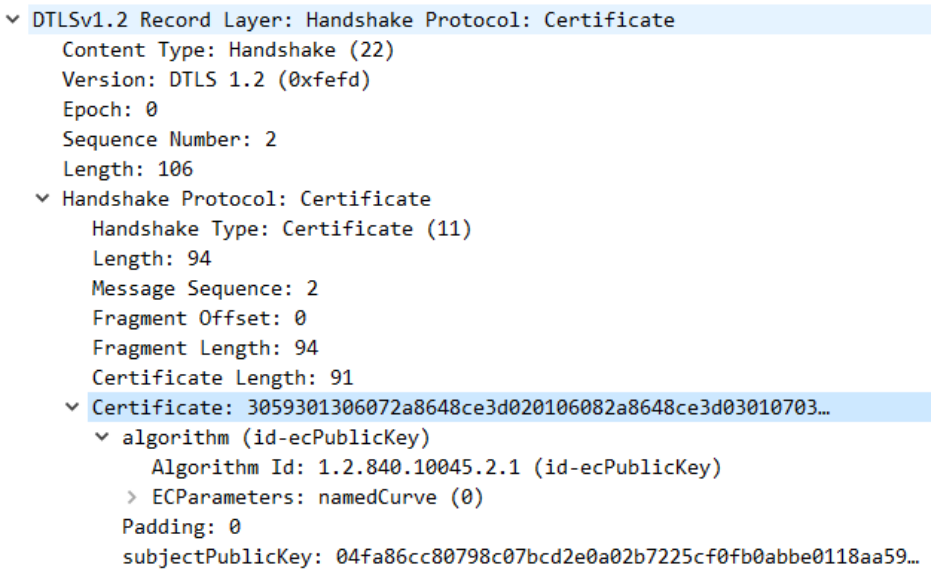

Figure 19: DTLS handshake Certificate message

Therefore, if either the client or the server wants to validate the other peer's certificate, they will have to obtain CA's certificate that issues the received certificate. It must be mentioned that the handshake has been executed with a default certificate provided by californium.

7.2.1.3 Functionalities

By default, the communication between the server and the client is configured to take place on localhost, however one can specify on the client side the following extra parameters, including the server address:

- Number of messages (application data) sent after the handshake is finished
- Those messages length
- Server IP address
- Server Port number

As far as certificate validations are considered, neither the program executed at server side nor the one at client side offer certificate validation. There is no signature verification and no revocation status checking.

7.2.2 Python3-DTLS

Another repository that offers DTLS implementation is *mobius-software-ltd/python3-dtls* [33]. It is an implementation of DTLS 1.0 or 1.2 in a Python environment based on the SSL module in Python's standard library. The main difference is that instead of creating a TCP socket an UDP socket must be created on its behalf.

7.2.2.1 Software requirements

Python3-DTLS works with Pyhon3 libraries, in particular, with Python 3.6 or higher. Furthermore, python scripts provided in this repository make use of python OpenSSL library, thus, its installation is necessary. On the one hand, Python OpenSSL module must be installed such as *pyOpenSSL 20.0.1* library whereas on the other hand, it is necessary to install Linux OpenSSL package. The latter must be OpenSSL 1.1.1 version or higher. Furthermore, there is no requirement for java jdk.

7.2.2.2 Implementation options

This repository provides a python script which plays the role of an echoing DTLS server which echoes every application message received from the client, thus, apart from executing the DTLS handshake, it also handles DTLS application protocol. With respect to the client side, another python script represents a DTLS client, which is configured to send messages to the server after the handshake is finished. Below, figure 20 shows an executed DTLS handshake:

Source	Destination	Protocol	Length	Info
127.0.0.1	127.0.0.1	DTLSv1…	253	Client Hello
127.0.0.1	127.0.0.1	DTLSv1…	86	Hello Verify Request
127.0.0.1	127.0.0.1	DTLSv1…	269	Client Hello
127.0.0.1	127.0.0.1	DTLSv1…	2256	Server Hello, Certificate, Server Key Exchange, Server Hello Done
127.0.0.1	127.0.0.1	DTLSv1…	175	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
127.0.0.1	127.0.0.1	DTLSv1…	308	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
127.0.0.1	127.0.0.1	DTLSv1…	81	Encrypted Alert
127.0.0.1	127.0.0.1	DTLSv1…	81	Encrypted Alert

Figure 20: Exchanged message during DTLS handshake using Python3-DTLS repository

In this scenario, there is no client certificate request. As it can be appreciated in figure 21, the server sends both its certificate and the certificate of the CA which issued its certificate. Then, in case the client wants to validate server certificate's signature, it is not necessary that it obtains the certificate of the CA that signed the certificate in question since it has already received it from the server.

```
▾ DTLSv1.2 Record Layer: Handshake Protocol: Certificate
    Content Type: Handshake (22)
    Version: DTLS 1.2 (0xfefd)
    Epoch: 0
    Sequence Number: 2
    Length: 1769
  ▾ Handshake Protocol: Certificate
      Handshake Type: Certificate (11)
      Length: 1757
      Message Sequence: 2
      Fragment Offset: 0
      Fragment Length: 1757
      Certificates Length: 1754
    ▾ Certificates (1754 bytes)
        Certificate Length: 870
      › Certificate: 308203623082024aa00302010202144216ac20491ff92e9… (id-at-countryName=ES,id-at-commonName=DTLSserver)
        Certificate Length: 878
      › Certificate: 3082036a30820252a00302010202147716f563c54c4566665… (id-at-countryName=ES,id-at-organizationName=PrimeKey,id-at-commonName=TestRoot)
```

Figure 21: DTLS handshake Certificate message

7.2.2.3 Functionalities

As OpenSSL library is used as the core of DTLS functions, many of its functionalities can be applied to a DTLS handshake, such as the followings:

- Specify the server IP address and its Port number.
- Configure it to execute only a DTLS handshake.
- Use blocking or non-blocking sockets.
- Validate certificate signature or chain.

Regarding certificate validations, there is a function used on the client side that allows to verify the received certificate signature as well as a certificate chain, if necessary. Moreover, it lets to enable or disable this functionality in the event that it is of no interest to carry them out. Finally, the client script does not offer certificate revocation status checking. However, as scripts are coded in python, fortunately, this programming language offers a great deal of libraries, many of them developed in python3, among which is an OCSP library.

Selection Criterion

Finally, the following criterion have been established in order to evaluate and select between the alternatives.

- **Dependencies**: It refers to required software that must be installed in order to implement each solution. The more conditions are imposed, the more problems one might encounter in terms of compatibility with other software tools in the project. Those software dependencies must also be evaluated taking into account their popularity and commonly used in the world of software.
- **Easy adaptation to project scenario:** It is helpful that the functionalities each repository provides adapt well and easily to project scenario. Moreover, it is also appreciated the ease to apply changes to the already existing code at each repository.
- **Certificate validation libraries for each programming language**: This criterion is imposed in view of the last validation implementation that must be done at client side. In the first scenario it does not mind using different programming language scripts on the client side and on the controller. However, since at the last step of the project a comparison must be made, all validation code must be implemented on the client side, then it is of interest to have both handshake and validation action in the same programming language. A glance at libraries used to carry out validation actions for each language program is recommended before evaluating this criterion. The language which has more available, is more used and ease to handle libraries for validation will be given more points.

	Scandium	Python3 DTLS
Dependencies (30%)	8	7
Easy adaptation to project scenario (35%)	5	8
Certificate validation libraries (35%)	4	8
TOTAL	5.55	**7.7**

Table 2: Evaluation of DTLS handshake repositories

Python3-DTLS repository will be used to implement DTLS handshake options. It can be appreciated at table 2 that it adapts well to project scenario since it executes a DTLS handshake with no requests of the client certificate, as validation of client certificate is not the aim of this master thesis since it is carried out by a powerful server. Moreover, there are more python libraries available to perform certificate validation actions.

7.3 Certificate Validation libraries

As Python3-DTLS repository is used, then, server, client and controller scripts will be coded in python. As already mentioned, the controller needs to carry out certificate validation actions. In order to do it, python libraries must be installed. Thus, in this section an analysis of certificate validation python libraries will be made, in particular, *pyOpenSSL,* and *cryptography* libraries will be analysed.

7.3.1 pyOpenSSL

PyOpenSSL is a python wrapper module around the OpenSSL library. It is compatible with python versions from 2.7 to 3.9. However, one must take into account that python 2.7 is already deprecated and it is not currently maintained. Furthermore, this library is the one Python3-DTLS repository uses to carry out the DTLS handshake.

It has a module called *crypto* which is a generic cryptographic module. It provides many methods and objects to handle X.509 digital certificates and Public Key Cryptography Standards such as PKCS7 and PKCS12. As previously mentioned, Python3-DTLS supports certificate signature verification, it uses OpenSSL to do that, thus, this alternative supports signature verification. The only drawback is that the function which offers this validation is part of the socket creation, then, it cannot be used to verify a signature outside this function.

pyOpenSSL support OCSP. It can have an open socket with a method; however, it is not possible to have an HTTP connection over that socket. Then another library or software would be required or developed.

7.3.2 cryptography

Cryptography is another python library that provides cryptographic recipes and primitives to Python environment. It is compatible with python versions higher than 3.7, including this one and PyPy3 7.2 or higher.

This library provides both objects and functions for high level recipes and low-level interfaces to cryptographic algorithms. On the one side, it supports Fernet, which is an implementation of symmetric encryption. On the other side, it has a great deal of functionalities to handle X.509 certificates and revocation requests and responses. Moreover, it also implements Certificate Transparency in case certificates need to be monitored.

In regard to certificate validation actions, cryptography provides some functions, which well combined, are able to verify the signature of a certificate. In addition, cryptography provides a module which allows to carry out the main important OCSP operations: load and create both OCSP requests and responses which are sent to or received from an OCSP server.

Selection Criterion

The following criterion have been established in order to evaluate and select between the alternatives.

- **Software compatibility:** It is highly important that installed libraries are compatible with already selected software to deploy the network and implement network modules.
- **Implementation of validation functions:** It is helpful that the library supplies modules or functions to carry out certificate validation actions, otherwise, the developer should invest time in codding them and checking them.
- **Dependencies and installation process:** This criterion refers to ease of installation process, in case the library depends on more modules or libraries, installation process becomes more tedious. Moreover, both the library and its dependencies must be compatible with other software of the project. Thus, the more dependencies a library has, the less likely to be compatible with all software of the project.

	pyOpenSSL	Cryptography
Software compatibility (30%)	9	9
Implementation of validation functions (50%)	0	10
Dependencies and installation process (20%)	7	7
TOTAL	4.1	**9.1**

Table 3: Evaluation python libraries

Cryptography library has been chosen. As table 3 shows, the main reason for the selection is to implement validation functions. Cryptography provides methods to perform demanded

certificate validation actions by the project whereas pyOpenSSL does not. The latter would require that the developer programs those validation methods, thus, it is much better to use cryptography library.

7.4 CA and OCSP responder

As already mentioned, an OCSP responder and a CA which issues digital certificates and, in turn, provides the OCSP responder certificates' status information is required in this project. Therefore, the last area to analyse is the implementation of an OCSP responder with its corresponding CA. In order to do that, two software implementations have been analysed: XiPKI and EJBCA. At this point, it is highly important to take software requirements into consideration before choosing alternatives to analyse since the new software and its dependencies must be compatible with the software other elements of the project use. E.g., Openssl version, java jdk, etc. Even though there are more available software to implement CA and OCSP responder service, there is no point in analysing them since they are not compatible with the other required software in this project. Thus, no matter how good they could be, they are far from compatibility with the project.

7.4.1 XiPKI

XiPKI stands for eXtensible sImple Public Key Infrastructure. It is available in *xipki/xipki* Github repository [35] and is a highly scalable and high-performance open source PKI, which offers a CA and an OCSP responder.

7.4.1.1 Software / Hardware requirements

XiPKI can be implemented in Linux, Windows and MacOS. Moreover, it can use any Java jdk from 8th to 13th version. As OCSP responders need to look for a certificate serial number in a database managed by a CA, a database is needed. XiPKI allows DB2, MariaDB, MySQL, Oracle, PostgreSQL, H2, HSQLDB. An application server is also necessary, Apache tomcat is used in XiPKI. Finally, with regard to hardware, any available hardware with at least 1GB of RAM and a processor of 900 MHz can be used for its implementation.

7.4.1.2 Installation

XiPKI Github repository provides information about its installation. Four main services must be installed. First a Database, later a CA server, next an OCSP responder and finally a Command Line Interface so as to be able to interact with the CA and OCSP responder. All of it is done by unpacking some zips and following steps provided in some files. This installation takes significant time, on the order of hours, since each service requires many diverse configurations and installation of other zips that depend on others and so on.

7.4.1.3 Functionalities

This software offers a great deal of functionalities, for instance it provides a complete CA. Some of the most relevant CA features are given below.

- It issues EdDSA, SHAKE and X.509v3 certificates.
- It is compatible with all databases mentioned in *6.4.1.1 Software requirements* section.
- Support for publisher for OCSP responder.
- Different public key types of certificates: RSA, DSA, SM2…
- Many certificate signature algorithms.

The advantage of having a CA with so many features is that very diverse certificates can be created according to different scenarios. Moreover, OCSP responder provides also several features such as supporting both unsigned and signed OCSP requests. Furthermore, it supports certificate status source published by EJBCA, DeltaCRL or by a customized Certificate Status Source. In conclusion despite its tedious configuration and installation process XiPKI is a complete open-source PKI that is particularly suitable for scenarios which require many and diverse interactions with a CA and/or OCSP responder.

7.4.2 Enterprise Java Beans Certificate Authority - EJBCA

EJBCA is a free software PKI Certificate Authority software package built using Java (JEE) technology. It performs certificate management, registration, enrolment and validation actions. It allows the user to configure a PKI in many different ways depending on one's needs. Since in this master thesis scenario, the main operation that EJBCA must perform is handling OCSP, it is going to be described how it offers OCSP service. Figure 22 shows EJBCA entities that form the OCSP responder. The first entity is a Root CA, at the other end of the path there is the Validation Authority that offers the OCSP service, thus, it will provide information about the requested validation state of a certificate.

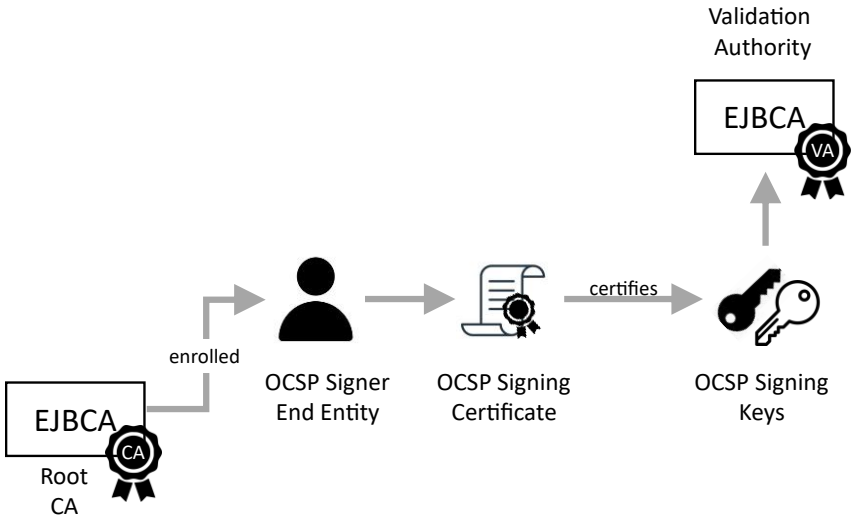

Figure 22: OCSP responder in EJBCA

Between the Root CA and Validation Authority, an End Entity is placed. It is a user of the PKI, which is not authorized to issue any certificate of its own, it requests certificates from the CA. In figure 22 scheme, the End Entity is an OCSP Signer, which is responsible for signing OCSP requests and responses. The OCSP Server is owned by a CA, which issues its certificates. Then, the OCSP server signs requests and responses using its private key, that is why there is a need of issuing certificates to the OCSP server as well.

7.4.2.1 Software / Hardware requirements

EJBCA requires some dependencies before its installation. *Java openjdk 8* is necessary. Furthermore, as the CA manages a database, then, a database client and server are required so as to handle it. EJBCA chooses Mariadb client and server, thus, these ones must be installed as well. EJBCA quick start guide [36] gives instructions on its installation. As far as hardware requirements are considered, EJBCA needs at least 1 GB of RAM and 1 CPU core of at least 1 GHz.

7.4.2.2 Installation

EJBCA software can be directly download from SourceForge repository [37]. Both a Community Edition and an Enterprise Edition can be downloaded. The latter, unlike the former, is not free of charge. All the same, community edition satisfies demanded functions by this project. EJBCA quick start guide [36] provides instructions about its installation. Its installation is pretty easy as the downloaded directory provides an installation script that installs and deploys WildFly application server and deploys EJBCA as well. The installation process takes just few minutes.

7.4.2.3 Functionalities

This software provides many functionalities as well, for instance it provides management of CA and sub-CA, which leads to the availability of diverse PKI functions. Some of the most relevant services EJBCA offers are the following:

- Issue User certificates as well as renewing both certificates and users
- Creation of Certificate Authorities
- Request of revocation or key recovery of a certificate
- Support for several Hardware Security Modules
- Certificate Transparency
- Support for OCSP responder

The latter, OCSP responder, can have many responder certificates, each issued by one CA. It can answer requests targeted at multiple CAs. Furthermore, it supports OCSP extensions that the administrator can define. All in all, EJBCA offers a PKI with a great deal of diverse functionalities. Its installation and configuration are simple tasks for the user and the interaction user-server is easy as well. EJBCA provides a complete documentation about different operations it supports, then, that makes much easier for the user to understand the software. It is well suited for any scenario which requires interactions with a CA and/or other additional services such as certificate revocation, key recoveries, etc.

Selection Criterion

Finally, in order to be capable of choosing between these two software implementations, they have been evaluated according to the following criterion:

- **Adaptation to project scenario:** It is desirable that the functionalities each software provides adapts well and do not pose a significant impediment or modification to the project scenario.
- **Installation process**: It refers to the time and complexity it takes to install CA and OCSP service. This is a fact to take into account in case it is required to install it again at some point of the project. Its complexity is also something important to consider. The more steps, complex and tedious an installation becomes, the more probability the administrator has to encounter with different errors, compatibility impediments or to forget an installation step.
- **User interface:** It refers to the interface the user can use to interact with the service as well as the available documentation about the services and functions the software supports.
- **Hardware requirements**: It is important to consider service hardware requirements since when talking about servers that carry out heavy operations, powerful RAM and/or CPUs are usually required. Extra memory and processing are always good to have. One

cannot forget that every software is installed on a VM of 2 GB of RAM and 2 CPUs at 1.8GHz.

	XiPKI	EJBCA
Adaptation to project scenario (30%)	8	8
Installation process (25%)	2	9
User interface (15%)	7	8
Hardware requirements (30%)	9	8
TOTAL	**6.65**	**8.25**

Table 4: Evaluation of software that provides OCSP and CA services

Despite both softwares offer similar features, as it can be appreciated in table 4 EJBCA has been chosen due to the tedious installation process of XiPKI which is likely to end in failure at some point. That is the reason for its low score. Moreover, EJBCA provides a better OCSP checking functionality and that is of great interest for this project.

8 RISK ANALYSIS

This section contains an analysis of all the risks that may arise during the development of this project. Every project is exposed to a set of risks, which must be analysed since they may have a considerable impact on it.

To begin with, it is necessary to identify every risk and describe them as well. Each risk must also be evaluated according to the estimated probability of its occurrence and the impact it would generate on the project if it were to happen. Finally, some contingency measures are defined with the purpose of minimising the impact of these risks.

8.1 RISK DESCRIPTION

After having analysed in-depth the nature of the project, three potential risks have been identified. A description of them is presented below. Furthermore, each risk is assigned an estimated probability of occurrence and an approximation of the level of impact that it would generate on the project.

8.1.1 R1: Unavailability of project members

A common risk present in almost all projects is that at least one of the people involved in it is not available for a period of time. This unavailability is usually due to any type of leave, such as sick leave among others, being on holidays, business trip, etc.

Therefore, it is significant to plan contingency measures in advance that avoid this period of unavailability to negatively affect the overall development of the project.

Probability: Medium

Impact: Medium

8.1.2 R2: Planification delays

Delays in any planification task or step is an important fact to consider. A task delay involves that all subsequent tasks which depend on the delayed one start later than arranged.

If deadline is approaching and there is still a lot of the project to be done, planification will be put at risk. This leads to trying to get everything done as quickly as possible, which may not go as planned and may end in failure. Otherwise, it should be assumed that the delivery date set for the project will not be met and new conditions may have to be agreed with the customer.

Probability: High

Impact: Medium

8.1.3 R3: Data loss or equipment failure

The loss of developed scripts and configuration files would mean having to start the project almost from scratch. As already mentioned, this project is developed in a Virtual Machine. VMs present a high tendency of crashing since their applications suffer from more outage than any other running on the host machine. After an outage, they are automatically closed, and present issues when trying to open them again or the VM does not even start. Not only will files be lost, but all the applied configuration and installed software will be lost as well in case the VM crashes. This would probably lead to the failure of the project due to lack of time.

Probability: High

Impact: High

8.1.4 R4: Previously undetected software incompatibility

Many different modules are used in this project. Thus, despite having analysed the requirements and dependencies each module demands, it is likely to encounter compatibility problems during its development. It becomes difficult to analyse all the requirements of a module before installing it since many dependencies appear during the module installation. Finding these incompatibilities could have severe consequences depending on the point of the project one encounters them. It can lead to a new alternative analysis of the problematic modules or in the worst case, start the project from scratch.

Probability: High

Impact: High

8.2 RISK PROBABILITY-IMPACT MATRIX

Once having identified and described each risk as well as rated them, each of them is represented in a probability-impact matrix, as shown in table 5.

		Probability				
		Rare	Unlikely	Moderate	Likely	Very likely
Impact	Trivial					
	Minor					
	Moderate			R1		
	Major		R2		R4	R3
	Extreme					

Table 5: Risk probability-impact Matrix

8.3 CONTINGENCY MEASURES

Finally, it has been determined the contingency measures to be taken for each risk in the event of their occurrence.

8.3.1 R1: Unavailability of project members

In order to mitigate this risk, the reason for the unavailability has to be evaluated as well as the role the missing member plays in the project. This project can be remotely developed since it is software-based. Therefore, if a member is not available due to a business trip or COVID-19 lockdown, it is not an obstacle as long as teleworking tools are available and online meetings can be carried out.

Nevertheless, in some cases such as holidays, as it is communicated in advance, project organization may be adapted to this unavailability carrying out tasks where the absent of the member does not imply any problem during that period, if possible. Otherwise, a substitute for the member must be found, which will also be the plan for indefinite absence or overnight sick leave.

8.3.2 R2: Planification delays

In order to avoid a significant delay in the planification, it is considerable to invest time in making a proper work planning. Every project stage should be over-scheduled, and it is recommended to set the project delivery date several days after the last stage deadline. This way, there would always be a few days of margin, which in case of having everything finished ahead of time, could be invested in possible project improvements.

In any case, it is considerable to have in the budget a provision for contingency so as to be used in case the project delay gives rise to cost overruns.

8.3.3 R3: Data loss or equipment failure

A simple and easy way to avoid this risk is to back up regularly project scripts, configuration files and exporting the working environment, the Virtual Machine. These backups must be stored on different devices and in the cloud as well since the used equipment in this project is a computer which can crash, be stolen or broken.

Furthermore, it is very convenient to note in a document everything that has been done per working day such as installed software or the reason of taken decisions among others. It is helpful in the worst-case scenario: having problems when importing the backed up virtual machine. This way, one knows exactly what to install and configure, which will save time and effort. That is why it is not enough to export the virtual machine, scripts and configuration files must be saved as well.

8.3.4 R4: Previously undetected software incompatibility

The best way to avoid this risk is to install the software required by all modules before starting to develop any of them. In other words, preparing the entire project environment in such a way that the following actions the developer has to perform are related to configuration, developing and deployment, since installation is already done. This way, in case software incompatibilities are found during installation process, it would not be a major problem to change any software module since the only action performed on each module has been just its installation process.

9 SOLUTION DESIGN AND IMPLEMENTATION

This section describes in detail the proposed solution in this project. In order to do it, first the design of the solution will be described. Secondly, the implementation of network architecture design and finally an explanation about network components and how they have been configured to offer the desired service will be shown.

9.1 DESIGN

Figure 23 shows how the designed IoT architecture looks like. It consists of five elements. An IoT device, the server it is going to send data to, a programmable switch placed between them, an OCSP server and finally the CA that has issued server's and client's certificate.

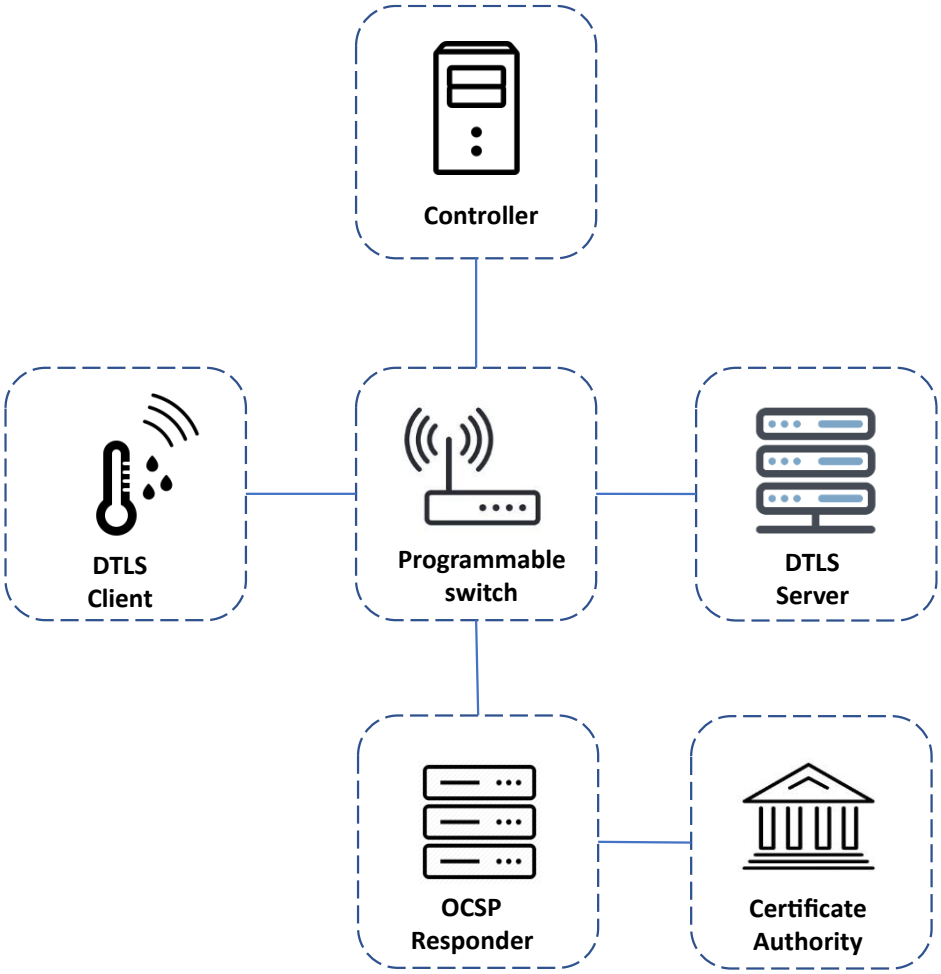

Figure 23: IoT scenario project design

- **IoT device:** It can be an industrial sensor, an intelligent vacuum cleaner or a voice assistant among many others. Its aim is to collect data from a user or from the environment and send it to a server, which is going to process it. However, information is not directly sent to it. During its path to the server, messages that contain this information are usually sent over a wireless network to a gateway and after that, to the

programmable switch. Anyway, as Mininet is going to be used in order to implement the solution, this gateway is not going to take part in the design. Data will be sent from the IoT client to the programmable switch.

- **Server:** The entity an IoT device sends information to. It collects received data from the IoT device and processes it.

- **Programmable switch:** This is a significant element in the design since it is responsible for carrying out certificate validation management. Even though it is called programmable "switch", it can behave as a router and handle 3rd level layer packets, as it happens in this project's scenario. The programmable switch has a control and a data plane, as already mentioned in section *2.5.1.3 P4 targets and architecture.* Control plane is managed by a controller, which is be explained on the next bullet point. Focusing on DTLS handshake, the analysed scenario in this project, the switch's data plane main functions are the following:

    - Identify the DTLS handshake message that contains the certificate/s.
    - Extract that/those certificates.
    - Send the certificates and the original DTLS handshake datagram, which contains the server certificate chain, to the controller.

- **Controller:** Every programmable switch's control is at least managed by one controller. As figure 23 shows, there is a link between the controller and the switch so as to control de switch's control plane. The controller must be responsible for carrying out certificate's validation actions. On the one hand, it will verify the received certificate signature. On the other hand, it will query an OCSP responder about certificates' status. Depending on the status of OCSP received responses from the responder and whether the verification ends in success, the switch will continue or stop the handshake process.

- **OCSP server/responder:** This entity is indispensable to validate the status of certificates. As previously explained in section *2.3.2.1 How OCSP works*, it receives OCSP requests about a certificate status and sends back its status in an OCSP response signed with its private key. According to the project model, the controller, is responsible for communicating with the OCSP server for checking the server's certificate revocation status. The OCSP responder will contact the CA which issue the server's certificate.

- **Certificate Authority, CA:** The CA is the issuer of both the Server's certificate and the OCSP responder's one. It maintains a database with all issued certificates' status and provides this information to the OCSP responder according to its requests.

9.2 IMPLEMENTATION

In order to explain the implementation of the design, first a general explanation about the deployed network is provided and finally each element that compose the network will be explained in detail.

First of all, network topology must be clear. Figure 24 shows how the network design is implemented in the working environment using selected tools in section *6.1 PROGRAMMABLE SWITCH SOFTWARE*. In comparison with the elements mentioned in *8.2 Design*, one can appreciate that there is a slight change. There is just one element called EJBCA instead of an OCSP responder and a CA. There is no problem, EJBCA is a tool which implements both the CA and the OCSP responder, then, it integrates both elements in one.

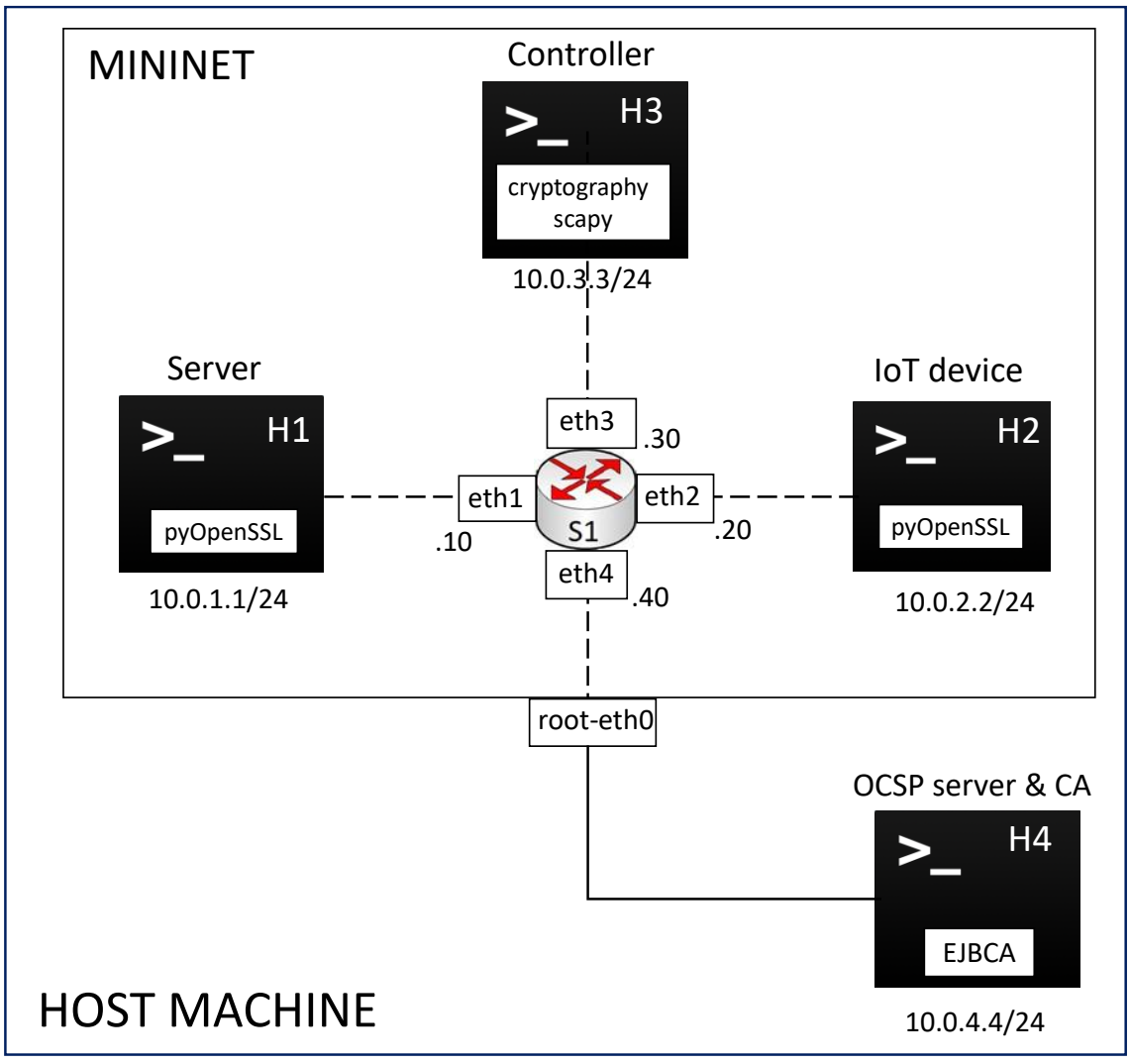

Figure 24: Network model

As the network model shows, EJBCA service is offered outside Mininet. Its reason is due to the fact that Mininet places each host in a separate namespace. Only the switch is placed in the root namespace. In a nutshell, hosts cannot perform every action that can be carried out in the host machine since they are not in the root namespace. Thus, they may also have limitations especially when running some host machine's programs such as EJBCA. Therefore, EJBCA is hosted in the host machine, where its service has no problem of being accessed. An interface that bridges Mininet with the host machine has been created to let this communication happen. Then, specific networking configuration has been applied in Mininet and the host machine so as to let figure 24 scenario work.

Firstly, necessary network configuration to implement figure 24 model will be described. After that, the required software installation, configuration and development of each element that constitute the system will be explained. Finally, how to run the DTLS handshake in that network model with everything already configured will be explained.

9.2.1 Mininet Network deployment

First, the network design must be deployed in Mininet.

9.2.1.1 Installation

As the VM provided by P4 language tutorials is used, it already has Mininet installed, thus, there is no need of any extra installation. In order to deploy a network in Mininet with the four entities, *basic* exercise of P4lang tutorials [32] has been used, in particular its *pod-topology*. This exercise's files have been modified according to this project's aim and network requirements. Extra files and scripts have been added to *basic* directory as well.

9.2.1.2 Configuration

In this section, it will be described how existing configuration files have been modified as well as how the new added ones look like. The following files have been modified:

- **run_exercise.py:** This file contains all functions that deploy Mininet network. Some parameters that are passed to these functions are specified in other configurations files that will later be described. Other relevant extra configuration that must be added to this script is provided. First, a port which will be used to bridge mininet to the host machine must be set. Its value will be taken from another configuration file: *topology.json*. At *__init__* function, the following line is added:

```
self.links = self.parse_links(topo['links'])
        # Add line below
self.outport = topo['switch_outport']
```

  After that, a new interface that links mininet and the host machine must be created and configured. At *create_network* function, the following code needs to be appended:

```
switch = defaultSwitchClass, controller = None)
        #Add the following
switch = self.net.switches[0]
root = Node('root', inNamespace=False)
intf = self.net.addLink(root, switch).intf1
ip = '10.0.4.4/24'
root.setMAC(intf=intf, mac='00:11:00:22:00:33')
root.setIP(ip, intf=intf)
```

- **topology.json:** As its name implies, it is a json file where network topology is declared. Each host IP and MAC addresses are specified as well as any configuration command that each of them needs to execute, such as their default gateway or ARP rules. Furthermore, switch configuration is also specified, however, as it is more complex, it will be explained in other two files: *s1-runtime.json* and *s1-commands.txt* that *topology.json* reads to specify switch's configuration. Finally, desired links between hosts and the switch are set.

Host	IP address	MAC address	Default gateway
H1	10.0.1.1/24	08:00:00:00:01:11	10.0.1.10
H2	10.0.2.2/24	08:00:00:00:02:22	10.0.1.20
H3	10.0.3.3/24	08:00:00:00:03:33	10.0.1.30

Table 6: Hosts' configuration

As table 6 shows, each host is placed in a different network, then the switch will forward packets at level 3. Below, *topology.json* configuration file is shown:

```
{
   "hosts": {
       "h1": {"ip": "10.0.1.1/24", "mac": "08:00:00:00:01:11",
              "commands":["route add default gw 10.0.1.10 dev eth0",
                          "arp -i eth0 -s 10.0.1.10 08:00:00:00:01:00"]},
       "h2": {"ip": "10.0.2.2/24", "mac": "08:00:00:00:02:22",
              "commands":["route add default gw 10.0.2.20 dev eth0",
                          "arp -i eth0 -s 10.0.2.20 08:00:00:00:02:00"]},
       "h3": {"ip": "10.0.3.3/24", "mac": "08:00:00:00:03:33",
              "commands":["route add default gw 10.0.3.30 dev eth0",
                          "arp -i eth0 -s 10.0.3.30 08:00:00:00:03:00",
                           "arp -s 10.0.4.4 00:11:00:22:00:33"]}
},
   "switches": {
       "s1": { "cli_input":"pod-topo/s1-commands.txt",
                "runtime_json" : "pod-topo/s1-runtime.json" }
   },
   "links": [
       ["h1", "s1-p1"], ["h2", "s1-p2"], ["h3", "s1-p3"]
   ],
   "switch_outport": {
        "s1" : {
          "port" : 4
        }
   }
}
```

In this case, switch outport is specified. This port will be the one which is used to bridge Mininet to the host machine through a switch interface. Paying attention to host H3 configuration commands, it presents one more ARP rule than the others. This rule is needed to let the controller, H3 communicate with EJBCA, H4. It is H4 responsibility to encapsulate the packet in a frame with ethernet destination MAC address the one specified in the ARP command.

- **s1-runtime.json:** Based on pod-topology of basic exercise, this file must be configured according to the following match-action rules shown in table 7.

Destination IP address	Destination MAC address	Switch Port
10.0.1.1/24	08:00:00:00:01:11	1
10.0.2.2/24	08:00:00:00:02:22	2
10.0.3.3/24	08:00:00:00:03:33	3
10.0.4.4/24	00:11:00:22:00:33	4

Table 7: Switch match-action rules

There must be specified the MAC address assigned to the Host H4, that now is the one specified in red.

- **s1-commands.txt:** As the switch is going to clone a packet, it can be specified the output port of the cloned packet. In this case, every cloned packet will be forwarded to port 3, the port connected to the controller.

Finally, Mininet network must be deployed. Once it is deployed, forwarding and ARP rules must be configured in host machine, where EJBCA is offered, so as to be capable of communicating with H3, the controller. Another important aspect is that there is usually a problem computing TCP checksum of packets sent from host machine to Mininet host. Therefore, checksum offloading at *root-eth0* interface should be disabled.

9.2.2 H4: Certificate Authority, OCSP responder – EJBCA

In order to get a digital certificate of an entity, a CA must be initialized. To do so, EJBCA free software PKI Certificate Authority software is used.

9.2.2.1 Installation and Configuration

Dependencies described at section *6.4.1.1 Software / Hardware requirements* must be installed. After that, it is time to configure EJBCA. It should be mentioned that until now, EJBCA service is not still installed since its dependencies must be first configured to let its installation happen.

Firstly, MariaDB configuration is going to be explained. It is worth mentioning that MariaDB is a fork of the MySQL database management system, thus, some configuration commands will refer to MySQL. The latter offers to set some basic configuration options such as allowing access only from localhost for the root account, set a password to that account, remove anonymous access, and remove the test database, which is accessible by all users. These options are configured

using *mysql_secure_installation* command. Focusing on this project scenario, as many tests are carried out during the development of the solution, and remote connections will be established, since EJBCA service is offered as a server in a network, it is better to disable those options.

Secondly, a database must be created. EJBCA quick start guide shows some commands that need to be adapted to this project. The only change that should be applied is the command which grants privileges to a user letting it access the database from localhost. It must be switched to the IP of the user or allowing its access from any IP providing a wildcard symbol. Last option is advisable due to the fact that OCSP requests will not only be sent from the controller, but also from the client when making tests at the end stage of the project.

After that, one must make the database listen in a specific IP address, or even better, listen in any IP address due to the reason mentioned in the previous paragraph. It depends on the IP address the service is offered at, in other words, the IP address of the EJBCA server in the network model. This configuration is applied editing */etc/mysql/mariadb.conf.d/50-server.cnf* file, the line *"bind-address"* must be switched from *127.0.0.0* to *0.0.0.0* if the database will listen on any IP address, otherwise the IP address where EJBCA service will be placed is specified.

Finally, EJBCA installation script: */ejbca_ce_7_4_3_2/bin/extra/ejbca-setup.sh*, must be configured with the name of the created database, the user, its password, and the IP address where EJBCA service will be offered in the network as well. As figure 24 shows, it should be deployed on 10.0.4.4/24 IP address. Afterwards, the final task is to run the script from the previous custom EJBCA directory where the unzipped folder was placed. This script will download Wildfly 10 and MariaDB database connector as well as installing everything to provide EJBCA service.

9.2.2.2 CA operations

There are two different ways to interact with EJBCA: from the GUI or using the terminal. In this case, every interaction has been executed using the terminal. In order to be capable of interacting with it, clientToolBox must be installed. The following actions have to be fulfilled so as to issue a server certificate:

1. **Create a CA:** EJBCA service is running, however, there is no Certificate Authority created. Thus, the first step is to create a CA which will issue the server certificate.

```
bin/ejbca.sh ca init --caname TestRoot --dn "C=ES, O=PrimeKey,
CN=TestRoot" --keyspec soft foo123 2048 --keytype RSA -v 365
--policy 2.5.29.32.0 -s SHA256WithRSA
```

   - **caname**: Name of the CA
   - **dn**: Distinguished name. It describes the identifying information in a certificate.
     - C: Country
     - O: Organization
     - CN: Certificate owner's common name
   - **keyspec**: Key speficiation for CA signing key and its size

- **keytype:** RSA, DSA or ECDSA
- **v**: Validity of the CA in days
- **policy**: PolicyId, imposed policy to the certificate.
- **s:** Signing Algorithm: SHA1WithRSA, SHA256WithRSA or SHA384WithRSA

2. **Add a new user to CA:** This user is the DTLS server, whose certificate will be issued by the previously created CA. These types of operations are carried out by a Registration Authority. The RA is an authority created to let users interact with the CA, thus, it allows to send certificate requests to a CA.

```
bin/ejbca.sh ra addendentity --username DTLSserver --dn "C=ES,
CN=DTLSserver" --caname TestRoot --type 1 --token PEM
```

- **username**: username for the new end entity → DTLSserver
- **caname**: CA issuing the certificate for the end entity
- **type**:
  - 0: Invalid
  - 1: End user
  - 256: Send notification
  - 512: Print User data
- **token**: token type for the end entity

3. **Create a Certificate Signing Request:** This request is created using openssl. One can specify several certificate fields such as Country, city, the entity and company name among others. Note that entity name must be the same as the already registered username in the CA. There is no need to run this command from *ejbca_install_dir/ejbca_ce_7_4_3_2/* directory as it is related with openssl. EJBCA does not take part in this step, openssl provides the command shown below to create files. One of that file is later sent to EJBCA as a request to create a certificate.

```
openssl req -new -newkey rsa:2048 -nodes -keyout server.key -
out DTLSserver.csr
```

This command creates two files, on the one hand *server.key*, which contains the server's private key and on the other hand, *DTLSserver.csr*, which is the CSR file. As soon as running it, the above command, certificate fields are prompted asking the user to fill them in. The server certificate has the following information:

- Country: ES (Spain)
- Region: Basque Country
- City: Bilbao
- Common Name: DTLSserver
- Password: pass

4. **Create Server's certificate:** Now, the created CSR in step 3 is sent to EJBCA. EJBCA creates a certificate based on this request. It is important to specify the path where CSR file was created:

```
bin/ejbca.sh createcert --username DTLSserver --password pass
-c /home/p4/DTLSserver.csr -f /home/p4/DTLSserver-cert.pem
```

   - **c**: path to the CSR file
   - **f**: path to the certificate that will be created.

6. **Get CA certificate:** Due to DTLS handshake used functions, CA certificate is sent along with the server certificate, therefore, it is necessary to obtain this certificate as well.

```
bin/ejbca.sh ca getcacert TestRoot -f /home/p4/ca-cert.pem
```

7. **Create an OCSP request:** In a normal operation of our solution, this OCSP request is created in a python script on the controller side. However, just to check that the OCSP responder works, and certificate creation and obtaining process has been successful, the following command lets one know about a certificate status.

```
openssl ocsp -issuer ca-cert.pem -cert DTLSserver-cert.pem
-req_text -url http://10.0.4.4:8080/ejbca/publicweb/status/ocsp
```

   If its status is good, not revoked, the response will look like the following:

```
Response verify OK
DTLSserver-cert.pem: good
This Update: Aug 30 19:18:22 2021 GMT
```

8. **Build server's pem file with private and public keys:** This step is required to perform the DTLS handshake, since this type of file must be provided by the server,

```
cat server.key DTLSserver-cert.pem > keycert.pem
```

9. **Revoke a certificate:** In order to carry out tests, EJBCA provides the option of revoking certificates. After revoking it, the same OCSP request as the one requested in the 6th step will be done. Therefore, one can see how the response of a revoked certificate looks like. First, in order to revoke a certificate, its serial number must be known, it can be known by an OCSP request of its status or asking the CA for its certificates that will expire within a specified number of days:

```
bin/ejbca.sh ca listexpired -d 366
```

   This command will prompt the subject's distinguished number and its serial number. Hence, once the serial number is identified, certificate's revocation can be requested:

```
bin/ejbca.sh ra revokecert --dn 'C=ES,O=PrimeKey,CN=TestRoot' -
s 3820C78FCF17F8A855DB6D6EBBB2B0A0D00B2CD2 -r 6
```

- **s:** certificate serial number
- **r**: revocation reason:
  - 0: Unused
  - 1: Key compromise
  - 2: CA compromise
  - 3: Affiliation changed
  - 4: Superseded
  - 5: Cessation of operation
  - 6: Certificate hold
  - 8: Remove from CRL
  - 9: Privilege withdrawn
  - 10: AA compromise

It is substantially important to pay attention at revocation reasons. The only revocation reason that lets the certificate be valid again asking for an un-revoking request is the reason identified by number 6. Therefore, it is the option used in this case. At this point, an OCSP response will look like the following:

```
Response verify OK
DTLSserver-cert.pem: revoked
This Update: Aug 30 19:32:58 2021 GMT
Reason: certificateHold
Revocation Time: Aug 30 19:27:22 2021 GMT
```

10. **Unrevoke a certificate** (If and only if its revocation reason is *certificate hold - 6)*.

```
bin/ejbca.sh ra unrevokeendentity DTLSserver
```

9.2.3 H1 & H2: DTLS Handshake – Python3, OpenSSL

The first step to follow in the implementation of the proposed solution is to configure the DTLS server, H1, and DTLS client, H2 letting them carry out a DTLS handshake.

9.2.3.1 Installation

In order to implement a DTLS handshake, *mobius-software-ltd/python3-dtls* [36] Github repository has been used, which works with python 3.6 or higher. It is moreover worth wondering if python 3.6 version will be compatible with other used libraries or if a higher version will be required. As already explained in section *6.3.2 cryptography* python 3.7 is required so as to be able to handle certificate validation libraries. Thus, Python 3.7 version is required. Once Python 3.7 is installed and set as default python version, OpenSSL library can be installed as well.

The library *pyOpenSSL 20.0.1* has been installed so as to be able to handle OpenSSL operations in python, which are required for the handshake execution.

9.2.3.2 Configuration

The repository provides scripts that simulate both the DTLS server and the client. These are *echo_seq.py* and *simple_client.py* respectively. The server simulates an echo server through a listen-accept-echo-shutdown sequence. The following changes have been applied to each script:

- **echo_seq.py:** The echoing function is removed as the important point is the handshake, then, only code related to the handshake will be kept. Moreover, the IP where the server listens is switched from *127.0.0.1* to *10.0.1.1*, the IP of H1 in the design. Finally, *python3-DTLS* provides an object used to create an SSL or DTLS connection. Below, an example of the creation of a DTLS connection object is shown:

```
scn = SSLConnection(
     sck,
     keyfile=path.join(cert_path, "keycert.pem"),
     certfile=path.join(cert_path, "keycert.pem"),
     server_side=True,
     ca_certs=path.join(cert_path, "ca-cert.pem"),
     do_handshake_on_connect=False)
```

  This object requires both the CA certificate and the DTLS server certificate. Moreover, a PEM file with the server's certificate and its private key must be provided. This repository passes some defaults certificates to the functions. Therefore, certificates created by the CA at *8.3.1.3 CA Operations* are provided to this function.

  - keyfile**:** keycert.pem generate in the 7th step of section 8.3.1
  - certfile: DTLS server certificate issued in the 4th step of section 8.3.1
  - ca_certs: CA certificate, obtained following the 5th step in 8.3.1

- **simple_client.py:** In the client script, the function that sends a message and the other that receives it are removed from the code since they do not take part in the handshake. As well as changed in the server's script, the IP the client connects to must be switched from *127.0.0.*1 to *10.0.1.1,* which is the server IP. Finally, signature validation is done by the controller, thus, it must be removed from the client script. This is done by removing the following parameter in the function that takes an instance of a socket:

```
s = ssl.wrap_socket(socket(AF_INET, SOCK_DGRAM),
cert_reqs=ssl.CERT_NONE)
```

  Therefore, as no parameters are provided, the function will not verify the certificate chain, this is responsibility of the switch, its controller verifies it.

9.2.4 S1: Switch data plane – P4

The switch data plane must be coded in some way it identifies the datagram that contains both server and CA certificates. This datagram, in turn, contains four DTLS handshake messages: Server Hello (SH), Certificates, Server Key Exchange (SKE) and Server Hello Done (SHD).

After identifying it, two actions are carried out. On the one side, the swich will clone that datagram and will send it through port3, where the controller H3 is connected to. On the other side, it will create a new datagram which contains only the certificates and will be transferred to the controller as well. The controller is also able to do every certificate validation operation receiving just the cloned datagram. However, sending the controller a datagram with just the certificates brings special benefits:

- It is more efficient processing the datagram at the switch data plane than doing it with at the control plane. The controller would do it with *scapy*, python, which processes at a higher level whereas a packet processing in the switch data plane is made at a lower level and is faster. *Scapy* is a python library able to copy or decode packets of many different protocols, send them, capture them and much more. No alternative analysis of scapy has been made since there is no library that does everything *scapy* does.
- Processing the datagram that just contains the certificates is an easy task for the controller to carry out, since it provides a module to get rid of everything except the payload. Thus, the switch takes computational burden off the controller.
- Therefore, the controller stores the original packet for further forwarding to the switch and obtains only the payload of the other datagram. With the payload, it can carry out validation actions.

9.2.4.1 Installation

As the VM provided by P4 language tutorials is used, it already contains required P4 tools installed. Then there is no need of extra software to program the switch data plane. The architecture used in this scenario is *V1model* and the target *Bmv2*.

9.2.4.2 Configuration

V1model architecture is explained in section *2.5.1.3 P4 targets and architecture*. This configuration section is divided according to the different elements that compose V1model. Therefore, first, the development of each element in the P4 program is explained and finally how the switch behaves joining all those elements is described.

- **Parser:** The parser has been configured as it is shown in figure 25. The aim of it is to identify the datagram sent by the server that contains the certificates and extract them. This datagram of interest contains four DTLS handshake messages: Server Hello (SH), Certificate, Server Key Exchange (SKE) and Server Hello Done (SHD). Extracting a header means removing data from a packet and storing it in defined header structures. Those

headers have been defined according to the type of information that must be stored and handled by other elements of V1model switch architecture. It is worth noting that the parser receives both the packet and some metadata about it, such as ingress port or instance type among others.

1. The parser extracts Ethernet header of every received frame. In the event of an IPv4 packet, it also extracts its IPv4 header.
2. Once IPv4 header is removed, the parser checks if the packet is received at the port where the DTLS server is connected to. One might wonder the reason for the latter condition. There are two grounds. On the one side, it behaves like a filter that analyses beyond IPv4 level only DTLS packets sent by DTLS server. It is only of interest to analyse messages sent by DTLS server since it is the endpoint which sends the message that contains its certificates, the message of interest. On the other side, original DTLS datagram which contains certificates will at some stage be sent to a controller. The latter, after some actions, will send it back again from its port. However, at that point, it is of no interest to analyse the packet again. All the same, it will be later explained in-depth. If this condition is fulfilled, the parser looks at IPv4 header's protocol field. It will check if its value indicates that the protocol used inside the IP packet is UDP. If both conditions are fulfilled, UDP header will be extracted.
3. In case UDP header is extracted, the parser looks how the following 8 bits looks like. In case a DTLS header is after UDP, those 8 bits belong to DTLS content type. As the type of the packet of interest is DTLS handshake, the parser will check if those 8 bits represent DTLS handshake content type. If so, DTLS header will be extracted.
4. Once DTLS header is extracted, the following 8 bits belong to the handshake type. If the 8 bits matches with Server Hello associated type, then, the parser extracts the completed Server Hello message.
5. The following 8 bits, as well, belong to another handshake type. If those bits represent Certificate type, three blocks are extracted. First, Certificate DTLS handshake header, secondly, server certificate and finally CA certificate.
6. The same rule is repeated, looking into the next 8 bits. If they match with Server Key Exchange type, SKE completed message is extracted.
7. Finally, checking the next 8 bits, if they correspond to Server Hello Done handshake type, the whole message is extracted.

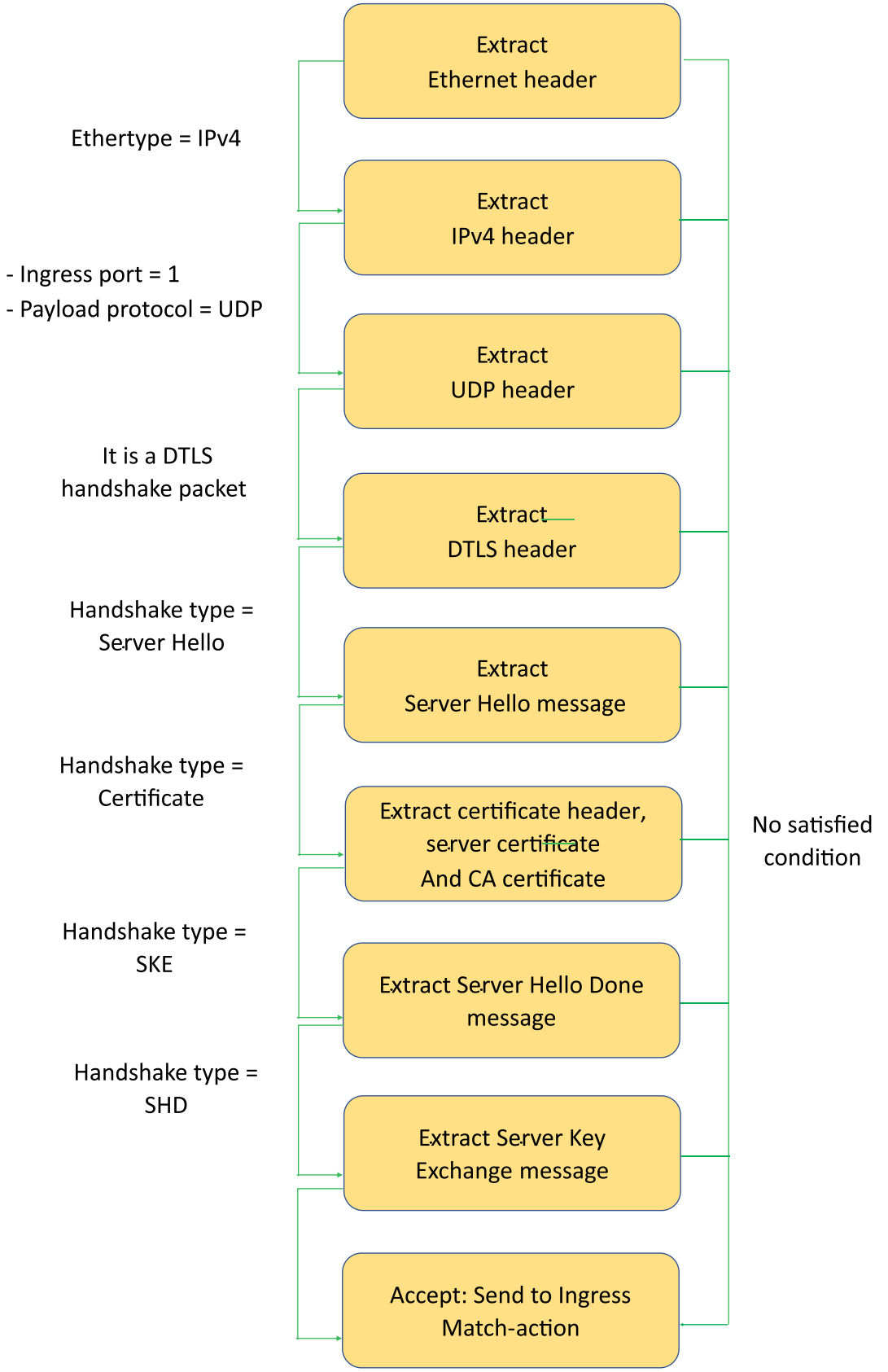

Figure 25: Parser states

- **Ingress Match-action pipeline:** After having the data of interest extracted and saved in headers, this ingress pipeline runs algorithms with them. According to the extracted headers and the original received packet's metadata, different modifications are applied to headers. Figure 26 shows the operations Ingress match-action pipeline applies to headers.

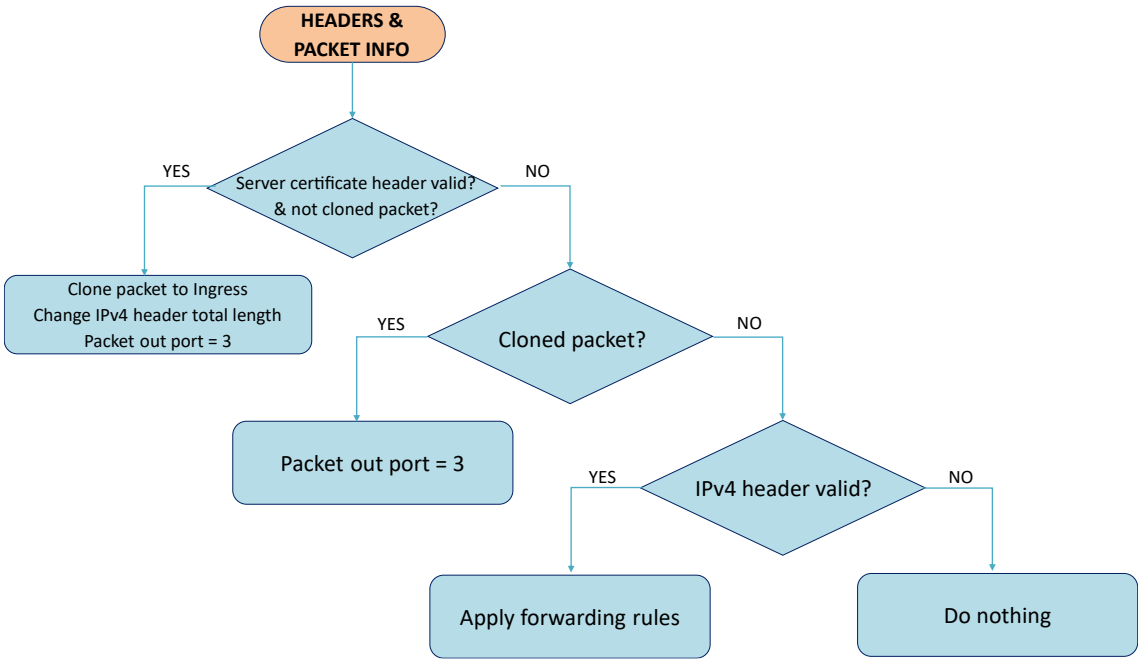

Figure 26: Ingress match-action pipeline state diagram

A brief overview of figure 26 is descripted below:

- If the packet of interest has arrived, the parser has already extracted certificates into a header. Therefore, if the packet has not be cloned yet, some actions are carried out. First, the packet is cloned. The reason for cloning it is that the switch is going to send two packets to the controller. One packet that only contains the certificates and other that is the original received one. The latter cannot be lost, then, this is the real reason for cloning the packet. As a new datagram is going to be created, IPv4 total length must be adapted to the new payload (certificates). Finally, it must be sent to the controller, egress port of the outgoing packet is set to port 3.
- If the first event does not occur, it is checked whether the packet is the cloned or not. If so, egress port of the packet is set to port 3 so as to be sent to the controller, H3.
- In case the previous condition is not satisfied it means that the packet is not the one of interest. Thus, if it is a IPv4 datagram, forwarding rules are applied according to its IP destination address. Otherwise, no action is performed.

- Clone to Ingress process: The action occurs at the end of the ingress pipeline. The cloned packet is a copy of the packet as it enters the ingress parser. It goes through parser states but without entering ingress pipeline. After that, it is cloned to a specified egress port, port 3 in this case. Moreover, it is worth mentioning that ingress port of a cloned packet changes. Both the cloned and the original headers are sent to the egress match-action pipeline. However, the cloned packet is first attended. After that it is sent to the deparser and extracted headers are appended to packet bytes that were not parsed.

- **Egress Match-action pipeline:** After the cloned packet has finished both the egress pipeline and the deparser process. Then, the original packet is processed by the egress pipeline. Figure 27 shows parser's actions. Its purpose is to invalidate all headers except the ones that are going to make the frame sent to the controller (Ethernet, IPv4, UDP and certificates) if it is the case of the message of interest (not cloned one). Otherwise, the egress pipeline will do nothing. Whenever data is extracted and stored in a header, the header is valid, the deparser only adds valid headers. So, at this point, the purpose of invalidating a header is to prevent the deparser from adding those invalid headers and thus create a packet with the valid ones: Ethernet, IPv4, UDP and Certificates headers.

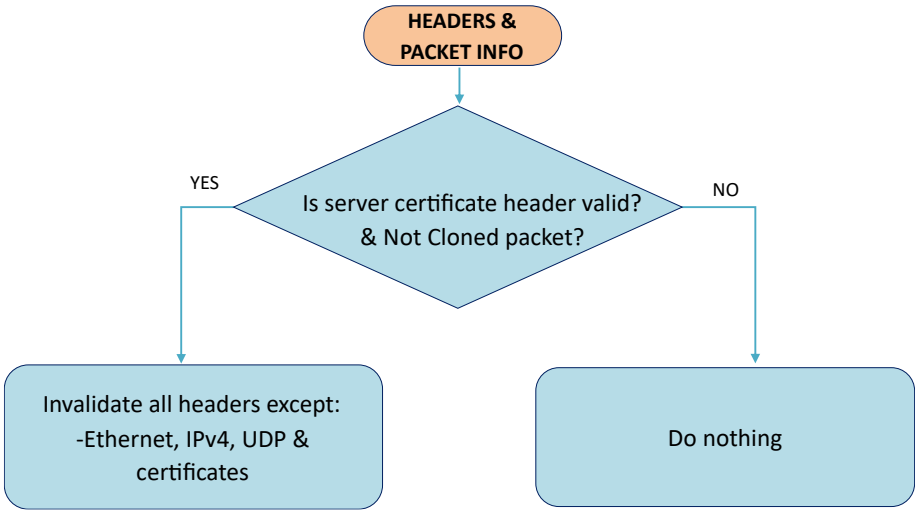

Figure 27: Egress match-action pipeline state diagram

- **Deparser:** It is worth mentioning that before the deparser headers, checksums are computed. Deparser will append all valid headers to the bytes of the packet that were not parsed.

After both the cloned and the original packet have finished all steps, they are sent to the controller. First the cloned one, as it was the first processed, and afterwards, the original packet, which now has been modified somehow its payload contains server and CA certificates.

9.2.5 H3: Controller – Python3 libraries

Finally, it will be explained how the controller has been programmed. A python script is run at H3. The script is developed in some way that it listens on an interface for incoming packets. It is supposed to receive both packets sent by the switch. The controller fulfils the following tasks:

9.2.5.1 Installation

In order to implement the controller, some pyhton3 libraries must be installed. On the one hand, *scapy* library allows the controller to handle packets. On the other hand, thanks to *cryptography* library, it is possible to carry out certificate validation actions.

9.2.5.2 Configuration/Development

As soon as executing python scripts, H3 starts listening on one of its interfaces. It is always listening, whenever a packet arrives it carries out some actions. Figure 28 shows this script behaviour:

1. If the datagram which contains Server Hello (SH), Certificates, Server Key Exchange (SKE) and Server Hello Done (SHD), the one cloned by the switch, is received, it stores it for later use.
2. After storing the previous datagram, if it receives the packet which contains both CA and server certificates, it gets the payload of the packet, in other words, obtains both server and CA certificates.
3. In case the controller got certificates, it validates server certificate signature. Otherwise, it does nothing, it continues listening on interface
4. If previous validation ends in success, the controller creates an OCSP request with those certificates and sends it to the OCSP responder, which is on H4, EJBCA. Then, it receives its response as well. If it ends in failure, it does nothing but continue listening for another packet on the interface.
5. In the event that server certificate's status is *good*, it means it is not revoked, then the controller sends the stored DTLS datagram back to the switch. In case certificate status is different from *good*, nothing is done, no packet is sent to the switch, the controller continues listening on the interface. Therefore, the handshake will be aborted after retransmission timers end at the client.

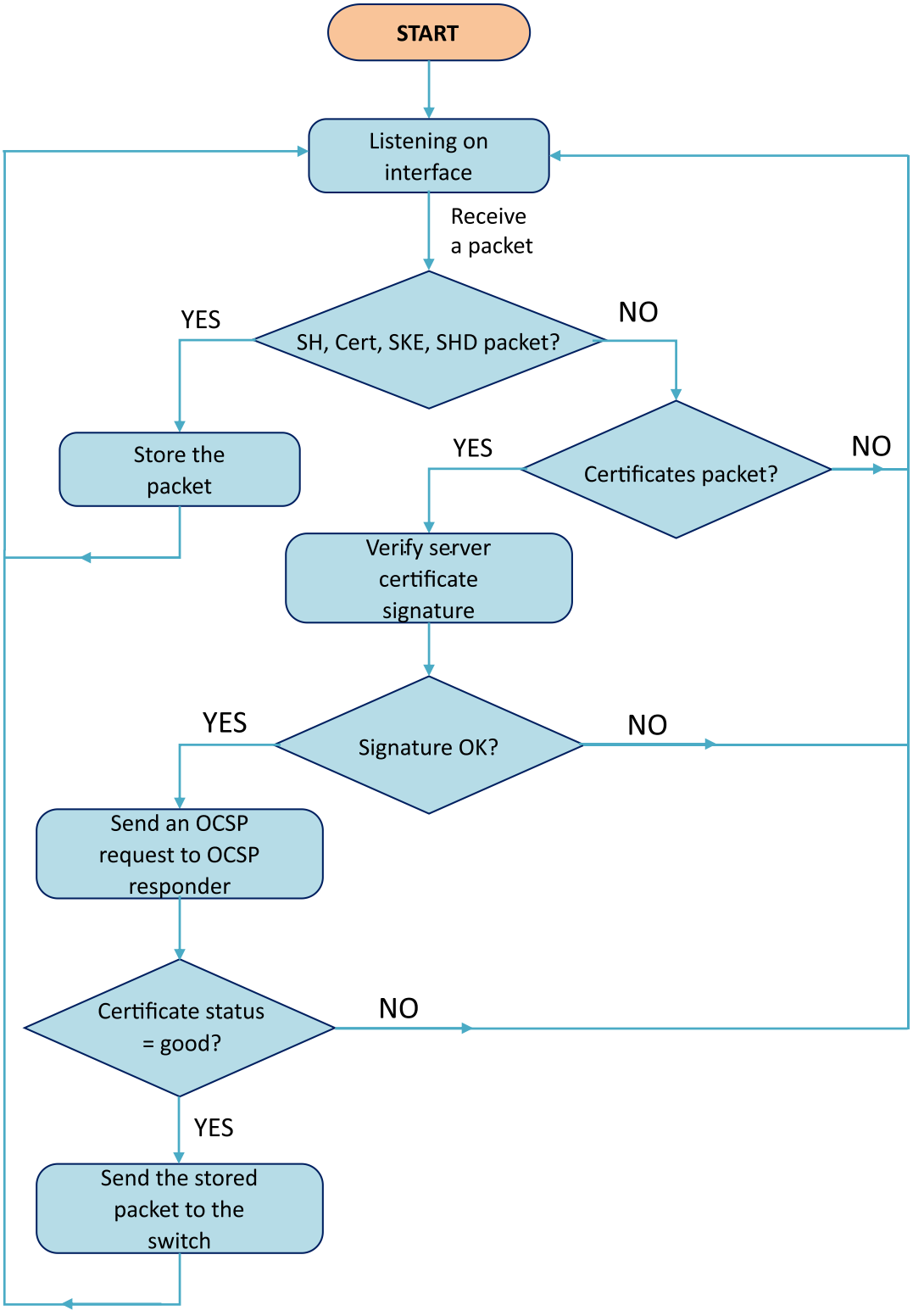

Figure 28: Controller actions state diagram

9.2.6 DTLS handshake start-up

Once everything is deployed, configured and ready to run a DTLS handshake. Three Mininet terminals should be opened: H1, H2, H3. In H1 and H2 terminals, one have to move to the directory where dtls library has been installed: */home/p4/.local/lib/python3.7/site-packages/dtls* and move as well to */test* directory so as to see DTLS server and client scripts. In H3, controller python script must be executed so as to make it listen for incoming packets and latter process them. Finally, one has to run *echo_seq.py* and *simple_client.py* scripts in H1 and H2 respectively and the DTLS handshake will happen. Below, DTLS handshake behaviour in this scenario is explained describing where packets are forwarded to, stored at, pass through or what else is done with them. To do so, DTLS handshake has been divided into the following four stages:

9.2.6.1 1st Stage:

This stage explains the behaviour of *Client Hello, Server Hello Verify* and *Client Hello* DTLS messages.

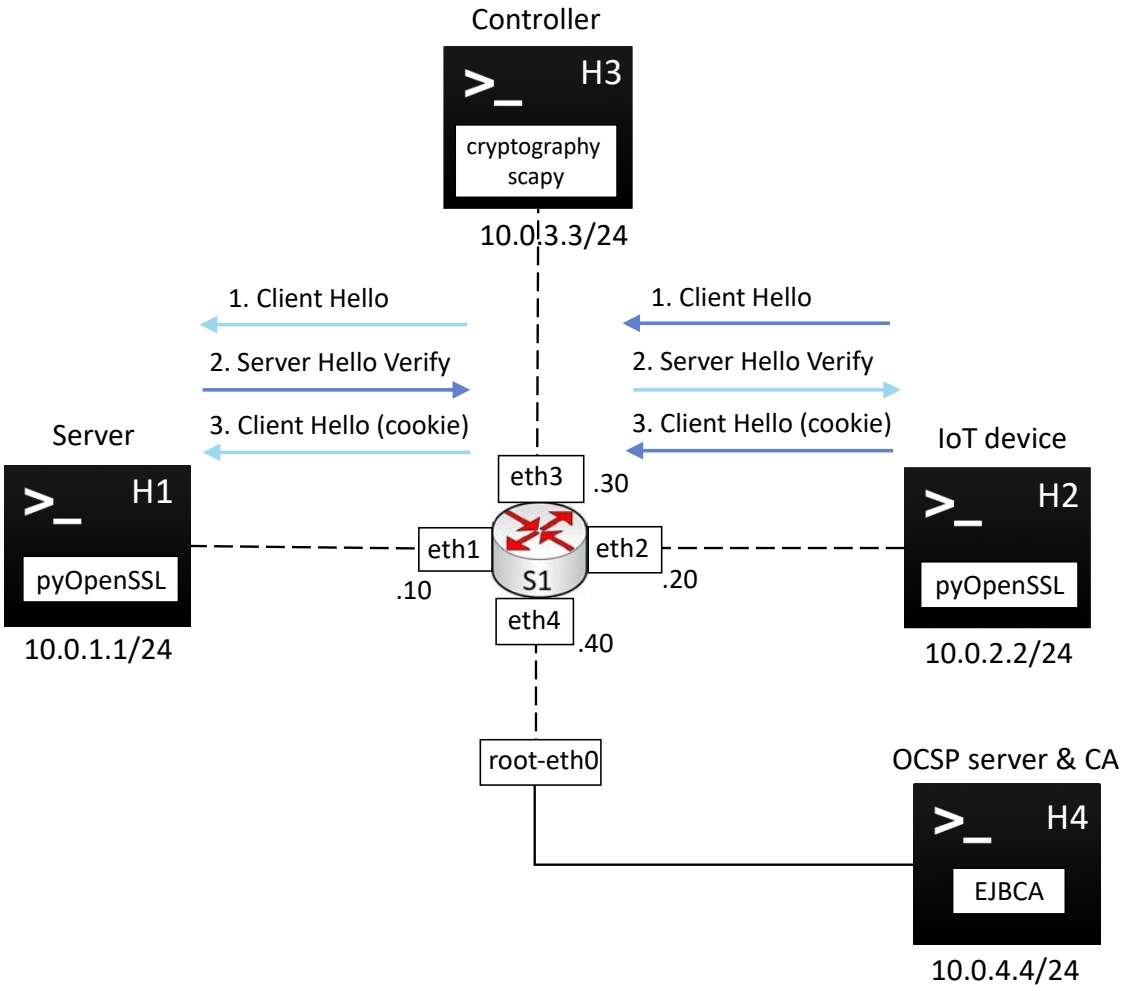

Figure 29: DTLS handshake first stage

As it can be appreciated in figure 29, at this first stage of the handshake, the switch and both the IoT device and the server take part in the communication. The programmable switch acts as a router, it forwards received datagrams. Therefore, as soon as the DTLS handshake is started by the client, the switch receives a *Client Hello* message, parses until IPv4 header and based on ethernet and IPv4 header, it changes its MAC address and forwards it to the destination MAC address, in this case, to the server. Same actions are applied to *Server Hello Verify,* in the opposite direction, and *Client Hello (with the cookie).*

9.2.6.2 2nd Stage:

At this second stage, how the switch processes the next datagram that the server sends is described. This datagram contains the following DTLS application messages: *Server Hello (SH), Certificate (Cert), Server Key Exchange (SKE)* and *Server Hello Done (SHD).* Figure 30 shows which parties are involved at this stage, the server, the switch and the controller.

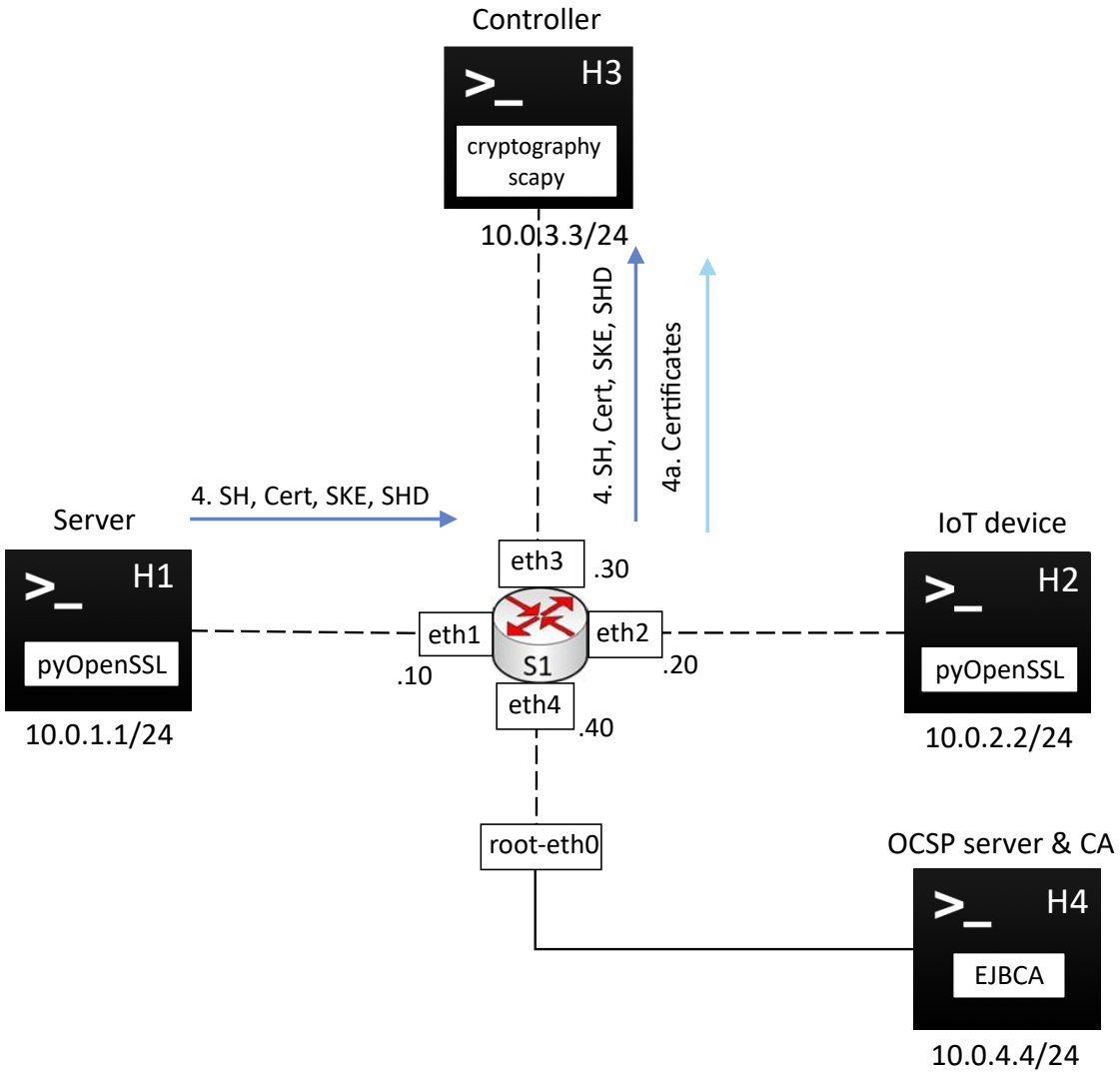

Figure 30: DTLS handshake second stage

When the server sends DTLS application message *4. SH, Cert, SKE, SHD* to the client. First the datagram reaches the switch, however, the switch does not directly forward it to the client. Switch's data plane realises that the datagram which contains the certificate has arrived as it has gone through all parser's state shown at figure 25. Therefore, the switch clones message *4. SH, Cert, SKE, SHD* and creates a new UDP datagram *4a. Certificates,* whose payload are only received certificates at message *4*, server and CA certificates. When both *4* And *4a* datagrams are ready, the switch forwards them to H3, the controller.

9.2.6.3 3rd Stage:

Now, it is time to interact with the OCSP responder and the CA. Figure 31 shows that this communication process involves the controller, the switch and both the OCSP responder and the CA.

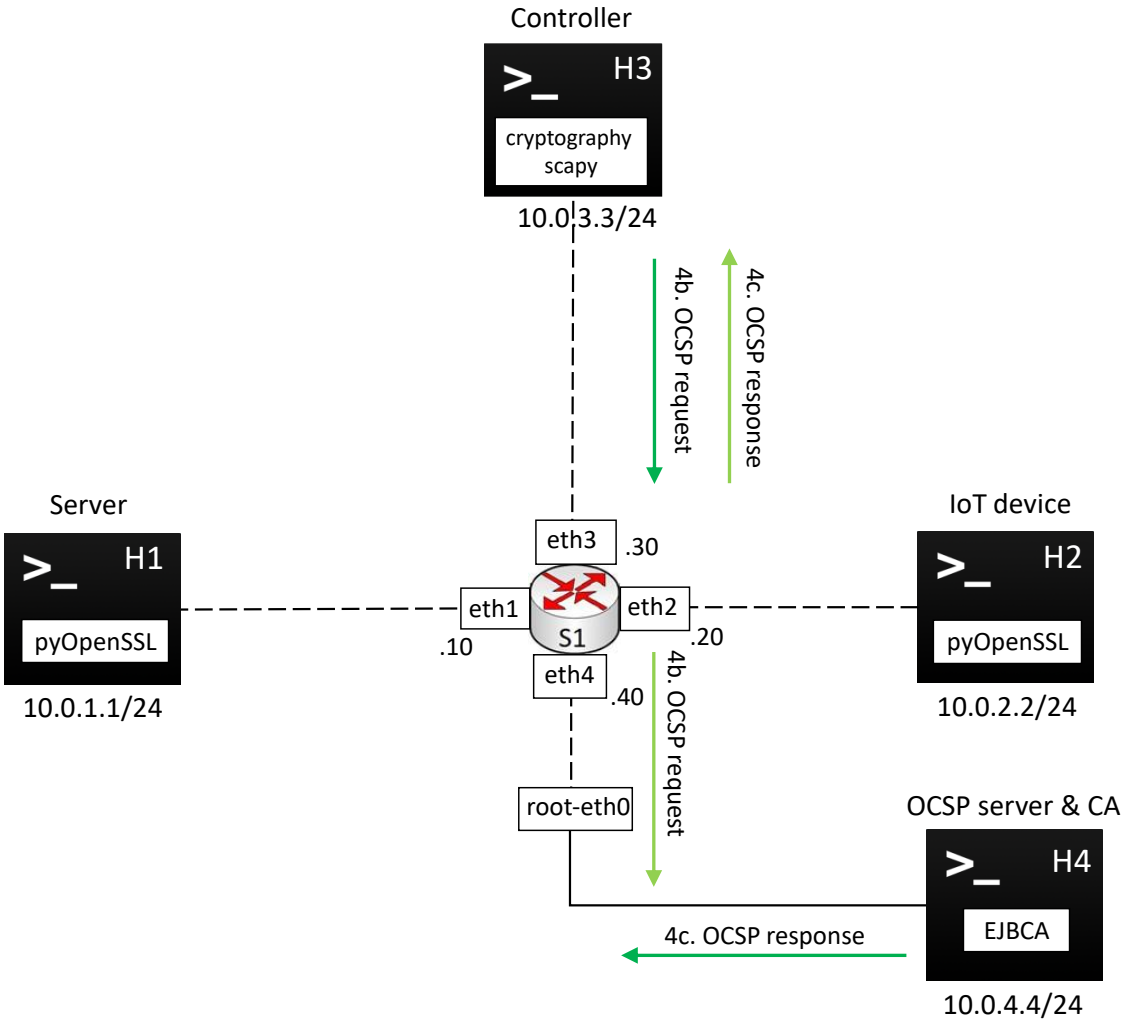

Figure 31: DTLS handshake third stage

Once the controller receives two datagrams at the previous stage, it does the following: It stores *4. SH, Cert, SKE, SHD* for later use and gets *4a. Certificates*'s payload. It verifies server certificate's signature and if it ends in success, the controller creates a new OCSP request about server's certificate: *4b OCSP request* and sends it to H4.

First, S1 receives the request, it applies forwarding rules to it, as it did on stage 1. Then, it forwards the packet to H4.

When H4 receives the OCSP request, it creates a response: *4c. OCSP response* and sends it back to H3. That response arrives to the switch, which forwards it to the controller.

9.2.6.4 4th Stage:

The last stage is shown in figure 32.

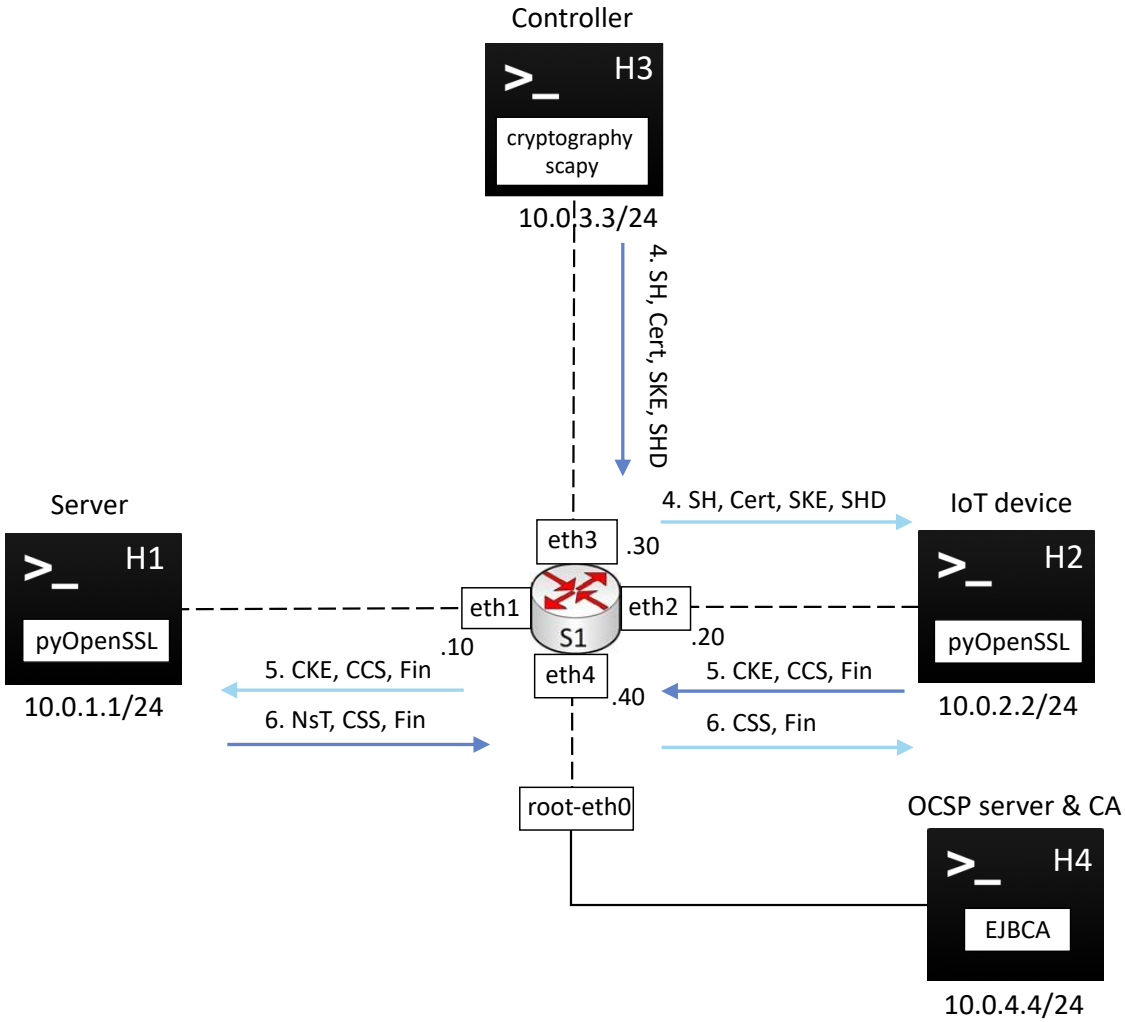

Figure 32: Last stage of the DTLS handshake

When the controller receives the response, *4c OCSP response* it looks at the status of the certificate, if the response said it is *good*, then, H3 sends the sorted message: *4. SH,Cert,SKE, SHD* to the client without any modifications, the DTLS message remains the same, as it was sent by the server. The switch forwards the datagram to the client, like a router would do.

As soon as the client receives *4. SH, Cert, SKE, SHD*, it creates the following messages *Client Key Exchange, Change Cipher Spec* and an encrypted *Finish* and sends them back in an UDP datagram: *5. CKE, CCS, Fin.* The switch forwards them to the server.

Now, at this final step, the server answers back with a new session ticket so as to start a new secure session and with the *Change Cipher Spec* and *Finish message* as well: *6. CSS, Fin*. Together with this final handshake message, the server sends a new session ticket, which is not represented in the picture above. This ticket contains the complete session state, including the master secret negotiated between the client and the server and the used cipher suite. Therefore, the handshake has ended in success.

To conclude, this P4 program can be easily adapt to different scenarios with regard to network requirements. Modifying just headers, parser states and/or match-actions will be enough to adapt this program to any requirement.

10 ANALYSIS OF RESULTS

This section presents obtained results once having implemented and run the solution of the project. An analysis of these results is carried out. In particular, a Functional Verification Test (FVT), whose aim is to check that the developed solution is working properly according to its design and specifications. In addition, several tests have been carried out so as to evaluate the performance of the system.

10.1 FUNCTIONAL VERIFICATION TEST – FVT

A FVT of the system is carried out. On the one side, the OCSP server is checked, just to make sure it answers back to OCSP requests. On the other side, FVTs based on network traffic capturing and analysis are performed. Packet capture is a term used in networking which means intercepting transferred data packets through a network interface in real-time without ceasing the communication. Captured packets are stored for a period, so that they can be analysed, archived, or discarded. To do so, traffic flow on different interfaces of the network model is analysed using *Wireshark*, a packet sniffer and analysis tool.

10.1.1 FVT1 -- OCSP server

It is significant to check if OCSP responder answers back correctly to an OCSP request about a certificate. First, a request of a valid certificate named *DTLSserver-cert.pem* is made. Below, both the OCSP request and its response show that OCSP responder actually works:

OCSP Request Data:
Version: 1 (0x0)
Requestor List:
Certificate ID:
Hash Algorithm: sha1
Issuer Name Hash: CEC083D6CE62141E8965D75AFC26DF894475C6CC
Issuer Key Hash: 939BA9B63909AFB7ADFE11C6918C1B48ED58BA4D
Serial Number: 21CA6ADF20EAD4C85FDE3039099722898BD49C79
Request Extensions:
OCSP Nonce:
041058559ECE366760C3247A9BE1BD696704

Response verify OK
DTLSserver-cert.pem: **good**
This Update: Sep 12 10:27:42 2021 GMT

Next, *DTLSserver-cert.pem* certificate has been revoked, stating the reason as *certificate hold,* and a new request about its revocation status is made. The corresponding OCSP response is shown below, thus, one can check that OCSP responder is working properly after these two different requests.

OCSP Request Data:
Version: 1 (0x0)
Requestor List:
Certificate ID:
Hash Algorithm: sha1
Issuer Name Hash: CEC083D6CE62141E8965D75AFC26DF894475C6CC
Issuer Key Hash: 939BA9B63909AFB7ADFE11C6918C1B48ED58BA4D
Serial Number: 21CA6ADF20EAD4C85FDE3039099722898BD49C79
Request Extensions:
OCSP Nonce:
041094625302453A6C5ABE81B95AAF8DCC65
Response verify OK

DTLSserver-cert.pem: **revoked**
This Update: Sep 12 10:50:28 2021 GMT
Reason: certificateHold
Revocation Time: Sep 12 10:49:39 2021 GMT

10.1.2 FVT2 – Traffic analysis

The aim of this FVT is to ensure that the system is sending desired data over the network. In order to do it, traffic flow is captured through switch's four different interfaces:

10.1.2.1 DTLS server – Switch

Having a look at figure 24 network model, exchanged frames between H1-eth0 and S1-eth1 are analysed. As figure 33 shows, the handshake is correctly performed, no packet loss. It can be appreciated a slight delay between frame 4th and 5th due to certificate validation operations. The server sends a DTLS datagram which contains the certificate, and it receives an answer from the client with a slight delay. Anyway, that delay is transparent and does not affect the handshake, as it is minimal.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.2.2	10.0.1.1	DTLSv1.2	253	Client Hello
2	0.001244226	10.0.1.1	10.0.2.2	DTLSv1.2	86	Hello Verify Request
3	0.003647213	10.0.2.2	10.0.1.1	DTLSv1.2	269	Client Hello
4	0.011769602	10.0.1.1	10.0.2.2	DTLSv1.2	2256	Server Hello, Certificate, Server Key Exchange, Server Hello Done
5	0.157979740	10.0.2.2	10.0.1.1	DTLSv1.2	175	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
6	0.158550667	10.0.1.1	10.0.2.2	DTLSv1.2	308	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message

Figure 33: Traffic flow between DTLS server and switch

10.1.2.2 DTLS Client – Switch

Traffic flow between H2-eth0 and S1-eth2 is captured and analysed. Figure 34 shows transferred frames. Transferred packets are the same as figure 33, as it should be since it represents the handshake between both two peers, thus, the switch should be transparent to both peers, and so it is.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.2.2	10.0.1.1	DTLSv1.2	253	Client Hello
2	0.001736729	10.0.1.1	10.0.2.2	DTLSv1.2	86	Hello Verify Request
3	0.001906484	10.0.2.2	10.0.1.1	DTLSv1.2	269	Client Hello
4	0.045779842	10.0.1.1	10.0.2.2	DTLSv1.2	2256	Server Hello, Certificate, Server Key Exchange, Server Hello Done
5	0.046359628	10.0.2.2	10.0.1.1	DTLSv1.2	175	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
6	0.047884575	10.0.1.1	10.0.2.2	DTLSv1.2	308	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message

Figure 34: Traffic Flow between DTLS client and switch

10.1.2.3 Controller – Switch

At this point, frames sent between controller and switch data plane are analysed, that means, between H3-eth0 and S1-eth3. As figure 35 shows, first, the original datagram that contains the certificates is sent to the controller, followed by another UDP datagram which only contains DTLS server and CA certificates. Once having received and having verified the signature, the controller sends an OCSP request to the OCSP server, frame 6. After that, the OCSP responder answers back with the OCSP response, frame 8. Finally, as the server certificate is not revoked, the controller sends back to the switch the original DTLS datagram, the first one received by the switch data plane. It is worth mentioning that TCP handshake messages are also exchanged between both parties. However, a filter has been applied so as to show only frames of interest in Wireshark.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.1.1	10.0.2.2	DTLSv1.2	2256	Server Hello, Certificate, Server Key Exchange, Server Hello Done
2	0.001806830	10.0.1.1	10.0.2.2	UDP	1815	28000 → 36409 Len=1748
6	0.008500856	10.0.3.3	10.0.4.4	HTTP	393	GET /ejbca/publicweb/status/ocsp/MHEwbzBtMGswaTANBglghkgBZQMEAgEFA
9	0.020801384	10.0.4.4	10.0.3.3	OCSP	178	Response
	0.040378155	10.0.1.1	10.0.2.2	DTLSv1.2	2256	Server Hello, Certificate, Server Key Exchange, Server Hello Done

Figure 35: Traffic flow between controller and switch data plane.

10.1.2.4 EJBCA – Switch

Finally, the traffic that flows between EJBCA and the switch is captured, which means, between EJBCA-eth0 and S1-eth4. Figure 36 shows the expected behaviour, an OCSP request and its corresponding OCSP response.

No.	Time	Source	Destination	Protocol	Length	Info
4	0.001321150	10.0.3.3	10.0.4.4	HTTP	393	GET /ejbca/publicweb/status/ocsp/MH
7	0.053967969	10.0.4.4	10.0.3.3	OCSP	178	Response

Figure 36: Traffic flow between OCSP responder and switch

10.2 SYSTEM PERFORMANCE

Once Functional verification tests have been successfully concluded, system performance must be evaluated. In particular, the time it takes to perform some actions is analysed. The following timings are measured:

- **t1**: Time it takes to perform the whole DTLS handshake
- **t2:** Switch data plane processing of (SH, Cert, SKE, SHD) DTLS datagram
- **t3:** Time it takes to check certificate revocation state.
- **t4:** Validation process: Time spent in verifying certificate signature and checking certificate revocation state.
- **t5:** Certificates' handling: Time it takes to obtain the certificates from the received packets and carry out its validation actions. In case of the switch's control plane (controller) it involves the time that elapses from receiving the datagram of interest (SH, Cert, SKE, SHD) until it is sent back after server certificate is validates. In case of the validation on the client side, it refers to the time the client takes to extract the certificates from the datagram of interest until it validates it.

In order to show that P4 program can be easily adapted to network requirements, timings have been analysed in two different situations:

- **Situation 1:** The entire DTLS handshake message that contains the certificates goes into a single UDP datagram.
- **Situation 2:** The sent chain of certificates is so large that fragmentation at network level is required.

For each situation, tests have been carried out in two different scenarios so as to evaluate the effect the proposed solution has on network performance:

- **Scenario 1:** Certificate validation is made at the switch's data plane and control plane (its controller). This is the solution developed in this project.
- **Scenario 2:** Certificate validation is made at the client's side. This is how an ideal and computationally powerful IoT client would perform certificate validation actions, no delegation on any other network element is made.

Several tests are carried out so as to analyse already mentioned timings, in particular thirty handshake tests per each combination of situation and scenario. Then, statistical measurements have been applied to obtained results for each timing so as to get a single value per timing. In particular, mean, variance, standard deviation and confidence Interval measurements have been evaluated.

10.2.1 Case 1: No fragmentation – Validation at the controller side

In this case, the message that contains the certificates goes into a single UDP datagram and certificate validation is done at the controller side, the switch data plane simplifies controller's actions. Table 8 shows previously mentioned statistical measures, in milliseconds:

	Mean (ms)	Variance (ms)	Standard deviation (ms)	Confidence Interval (ms)
OCSP	3,560	10,974	3,257	2,362-4,836
OCSP + signature	3,667	10,972	3,257	2,429-4,902
Switch processing	1,3	21,724	4,583	1,126-1,474
Certificates handling	24,175	18,861	4,269	22,553-25,796
Whole handshake	55,855	340,2	18,135	48,972-62,742

Table 8: Time measurements in milliseconds, controller validation, no fragmentation

Despite the operations performed by the switch, already explained at *9.2.4 S1: Switch data plane – P4*, as they are carried out in the data plane, the time is of units of milliseconds. Therefore, it can be claimed that switch's data plane is really fast in performing actions. It can be appreciated that most of the time it takes to perform the handshake is added by the controller since it must handle two packets, storing one of them and extracting certificates from the other as well as validating the server's certificate and sending back again to the switch the original DTLS handshake message that was stored.

10.2.2 Case 2: No fragmentation – Validation at client side

Certificates are sent in the same UDP datagram as well. However, now, certificate validation checking is made at the client side. Thus, the switch only forwards packet, it does not handle certificate's message in a different way from the others, then, its processing time has not been measured as it is minimal, in terms of microseconds. Table 9 shows the following time measurements:

	Mean (ms)	Variance (ms)	Standard deviation (ms)	Confidence Interval (ms)
OCSP	3,354	5,136	2,229	2,525 - 4.186
OCSP + signature	3,458	4,969	2,193	2,646 - 4,276
Certificates handling	4,645	9,546	3,039	3,511 - 5,778
Whole handshake	17,266	42,787	6,435	14,867 - 19,666

Table 9: Time measurements in milliseconds, client validation, no fragmentation

Validation times are similar to the ones shown at the previous table. Certificate handling is lower. Regarding the handshake, it is lower, as expected, due to be fact that there is neither extra switch processing of the message of interest nor extra hop before arriving to the client. In the previous scenario that message is processed by the switch, sent to the controller (one hop) and processed by it, then sent again to the switch (two hops), and finally, forwarded to the client (three hops). Moreover, there is no controller, which is the one that more delay introduces with its actions.

In both cases, a minimal difference between the time it takes to ask the OCSP responder about the status of a certificate and get its corresponding response, and the time It takes to perform the whole validation (OSCP + signature) can be appreciated. Therefore, it claims that revocation checking actions involve much more time and cost than verifying certificate's signature. The latter, as it is a simpler task, it is usually implemented by DTLS handshake whereas the first one is not. The main delay on the handshake is due to the actions of handling certificates by the controller and the hops the message of interest must do before arriving to the client.

10.2.3 Case 3: Fragmentation – Validation at the controller side

The certificate chain is so large that the UDP datagram which should contain the DTLS message where the certificate goes in, must be fragmented. Certificate validation is done at the controller. In this case, the UDP datagram has been fragmented into two fragments, then, the switch has one more datagram to handle. Obtained time measurements are shown in table 10.

	Mean (ms)	Variance (ms)	Standard deviation (ms)	Confidence Interval (ms)
OCSP	2,853	3,502	1,840	2,154 - 3,552
OCSP + signature	2,906	3,500	1,840	2,207 - 3,604
Switch processing	4,133	1,361	1,147	3,700 - 4,569
Certificates handling	34,313	32,259	5,584	32,192 - 36,434
Whole handshake	76,621	295,427	16,900	70,203 - 83,039

Table 10: Time measurements in milliseconds, controller validation, fragmentation

As it can be appreciated in table 10, validation times are similar to previous cases and switch processing slightly higher since the switch needs to perform actions in two datagrams instead of one as it happened in the first case. The same happens at the controller side, it has to handle more datagrams, then more actions to validate the certificate, which ends in an increase of time in certificates handling, and that is the main reason for the increase in handshake time.

10.2.4 Case 4: Fragmentation – Validation at the client side

As well as in the previous case, the UDP datagram must be fragmented. The validation, now, is made at the client side. Table 11 shows obtained results:

	Mean (ms)	Variance (ms)	Standard deviation (ms)	Confidence Interval (ms)
OCSP	4,092	7,046	2,609	3,101 - 5,084
OCSP + signature	4,156	7,026	2,606	3,165 - 5,145
Certificates handling	5,606	13.,463	3,578	4,247 - 6,966
Whole handshake	19,713	224,352	14,726	14,120 - 25,306

Table 11: Time measurements in milliseconds, controller validation, fragmentation

Both OCSP and complete validation time are similar to previous cases. Regarding handshake time, the involved time is low since there is no extra hop for the fragmented messages that contain the certificate and the switch just need to forward one more message in comparison to cases 1 and 2, which is not meaningful as forwarding time is minimal.

10.2.5 Comparison of the results

Some graphics are shown below so as to compare the mean of each timing per evaluated case.

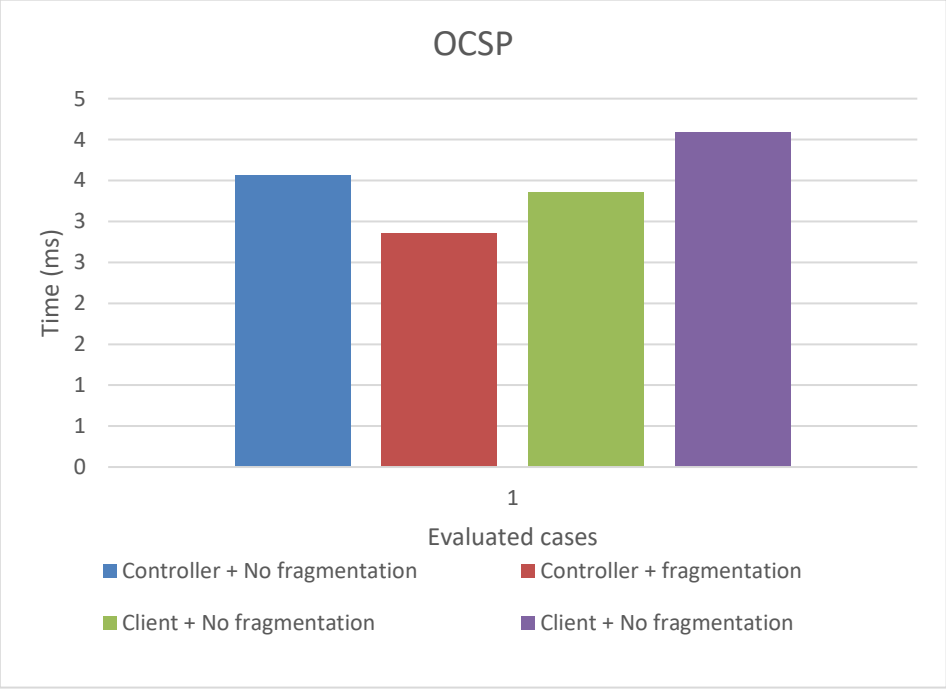

Figure 37: Mean of certificate revocation checking time of each evaluated case

As figure 37 shows, all cases present similar certificate revocation checking times. Results should be similar since the query to OCSP server is made using the same function in python and the number of hops involved in the query are the same in all four cases. The switch forwards both the OCSP request and response, which go in HTTP messages, then, over TCP. In all four cases, the OCSP request is segmented by TCP into two packets, then, the number of sent packets is also the same in all cases.

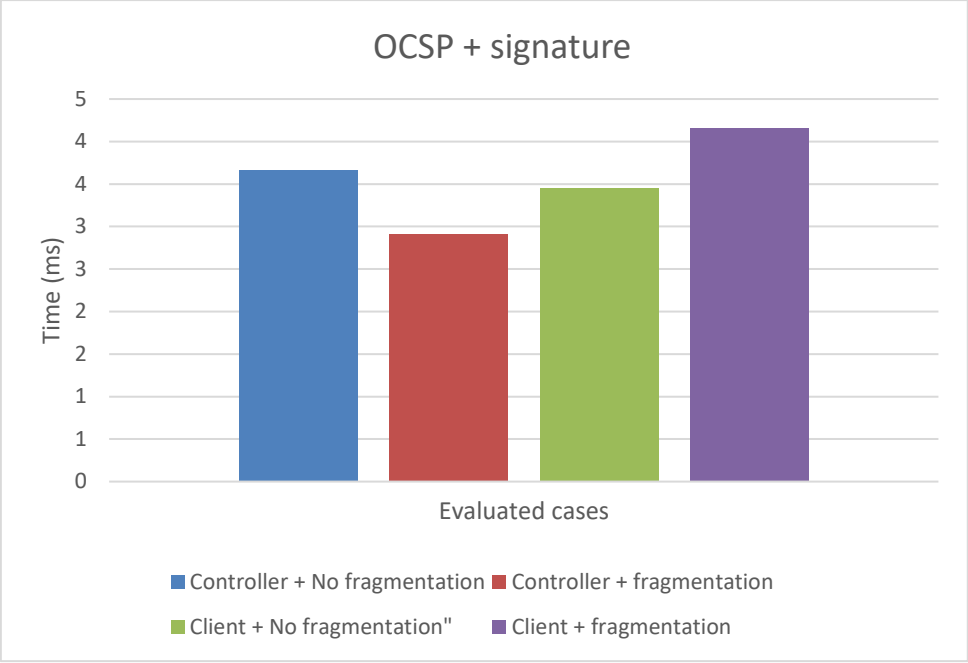

Figure 38: Mean of certificate validation time of each evaluated case

As figure 38 shows, revocation checking, and signature verification time are also similar in all four cases due to the reason mentioned in the previous paragraph.

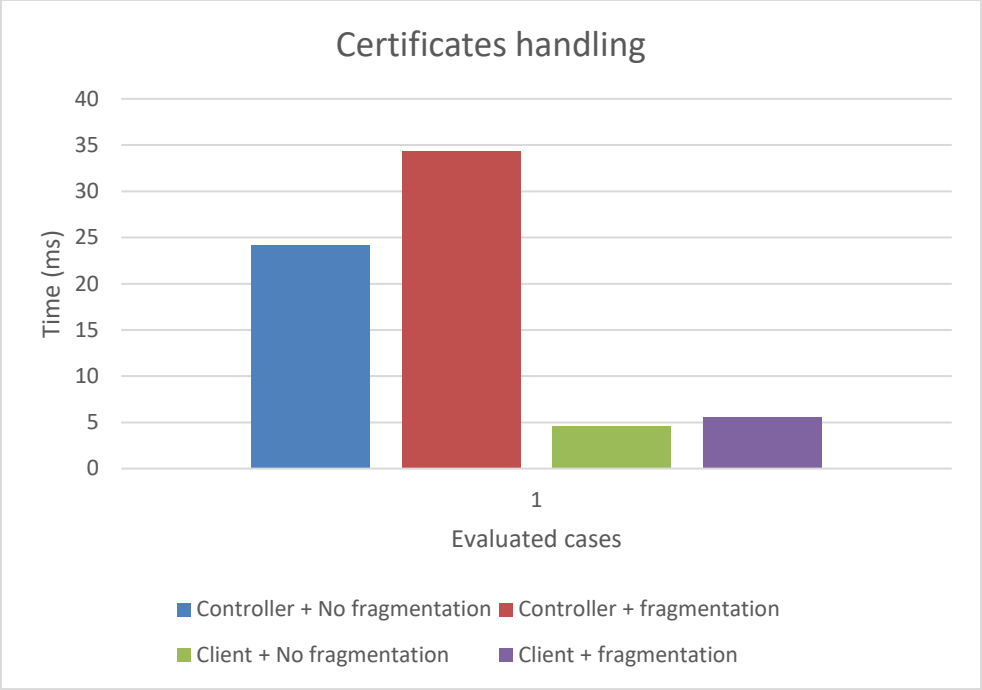

Figure 39: Mean of Certificates handling time of each evaluated case

Figure 39 shows the time differences regarding certificates handling. It is the main time difference between doing the validation at the controller side and doing it at the client side. However, despite it seems to be a high difference, it is not so much actually.

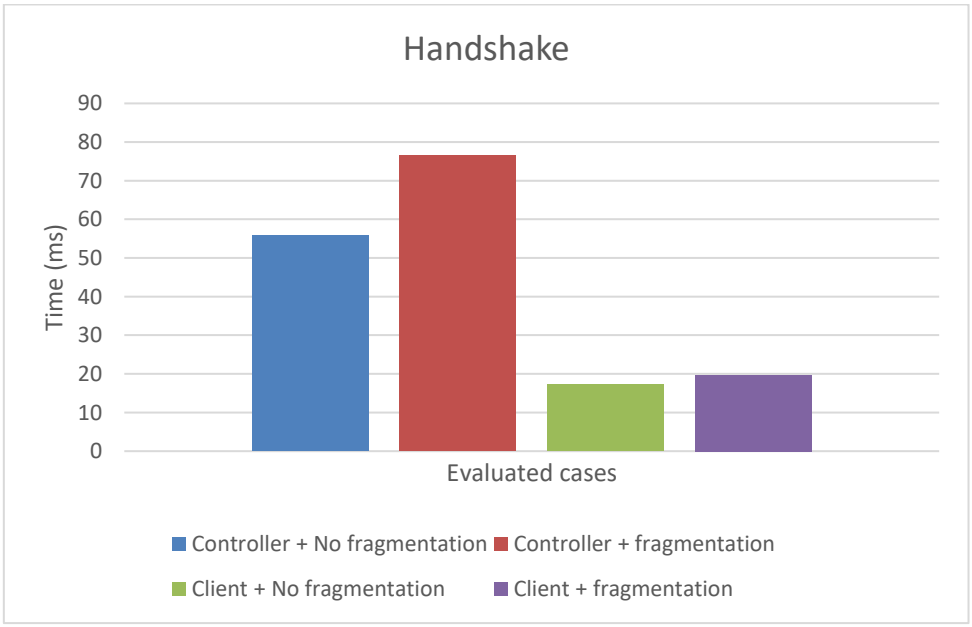

Figure 40: Mean of Certificates handling time of each evaluated case

Finally, the time it takes to perform the whole handshake is very different between all evaluated cases. As figure 40 shows, in the thrid and fourth cases it is the IoT device which performs certificates' validation, in contrast with first and second cases, where the controller carries out this validation with the switch's help. Obtained results are as expected, since the cases where the controller and the switch perform certificate validation actions involve extra hops of the DTLS message that contains the certificate, as well as extra actions to carry out on the switch data plane and control plane.

Now, the effect that the implemented solution brings to the DTLS is evaluated. For that purpose, Validation at controller side and validation at client side are compared in each situation.

- **Situation 1:** The entire DTLS handshake message that contains the certificates goes into a single UDP datagram. In this case the extra time that the implemented solution adds is of 39ms, which is not something meaningful. The DTLS handshake is not interrupted and none of the remote peers of the communication notices what is happening with the DTLS message that contains the certificate. Certificate validation is done in a transparent way for both peers.
- **Situation 2:** The sent chain of certificates is so large that fragmentation at network level is required. Now, the extra time that this project's solution adds is of 57ms. It is mainly due to fragmentation at network level. Both switch's data plane and control plane need to carry some more actions and the validation cannot be done until the last fragment has arrived. However, it is the control plane which most delay adds, data plane actions are performed really quickly despite performing twice as much as in situation 1. Thus, it is understandable that DTLS handshake time is increased. However, 57ms is nothing to worry about.

11 WORK METHODOLOGY

This section presents a planification foreseen for the development of the project. Work packages (WP) involved in this work are described. To do so, first, the work team is explained. Then, different phases and tasks the work has been divided into are described. Finally, a Gantt diagram showing an estimation of the time period each task involves is presented.

11.1 WORKING TEAM

Table 12 shows the working team members responsible for carrying out the project.

Code	Name	Responsibility	Played role
I1	Jasone Astorga	Senior Engineer	Director of the project
I2	Eduardo Jacob	Senior Engineer	Co-director of the project
I3	Gloria Zufiaurre	Junior Engineer	Developer of the project

Table 12: Working team

11.2 WORK PACKAGES

This section defines all work packages involved in carrying out the project. The following six WPs have been identified.

11.2.1 WP1: Problem statement

The first work package consists of clarifying the scope of the project with both director and co-director. It involves making a proposal of the project identifying its benefits as well as its feasibility. This work package consists of four tasks, which are described below:

- **T1.1: Study and familiarization with technologies:** First a study of existing technologies that the project is based on is needed. Furthermore, it is advisable that the junior engineer gets started with those technologies following learning tutorials they provide for instance.

  Task duration: 40 hours

- **T1.2: Objectives and requirements specification:** The junior engineer together with its supervisors, senior engineers, specifies the scope and requirements of the project. This, in turn, involves identifying the benefits of the project as well as its feasibility.

  Task duration: 10 hours

- **T1.3: Analysis of state of the art:** This task involves a study of related work. In a nutshell, projects and publications whose aim resembles the scope of the project.

  Task duration: 36 hours

- **T1.4 Analysis of Alternatives:** Once objectives and requirements are defined, design and implementation alternatives to carry out the project are evaluated. An assessment of Virtual Environment and the required modules to implement desired applications is made.

  Task duration: 50 hours

11.2.2 WP2: Solution design and working environment installation

This work package's purpose is to design the solution, which means a design of the network model and the applications it runs, as well as installing all required software so as to ensure all necessary tools are installed for its later implementation.

- **T2.1 Network architecture design:** The network architecture is designed taking into account Mininet network emulator as well as its features and restrictions running desired applications.

  Task duration: 100 hours

- **T2.2 Software installation:** All required software to develop the project is installed at this stage. This way, one can check that each software is compatible with all the others.

  Task duration: 30 hours

11.2.3 WP3: Implementation of a basic network

As far as project implementation is concerned, first a basic network is deployed which does not offer certificate validation. It consists of a DTLS client and its respective server and a programmable switch with its controller. Its aim is to implement a DTLS handshake between two remote peers separated by the switch and to give the controller enough information to let it validate a certificate at the next work package.

- **T3.1 Network architecture configuration:** In order to be capable of providing connectivity between network elements, both network and each element of it is configured before the network is deployed. In this task, on the one side, network general aspects such as topology creation are setup. On the other side, network elements are configured. Hosts' layer 3 and 2 configuration are set, and switch data plane is programmed to handle packets following the requirements specified in the solution.

  Task duration: 30 hours

- **T3.2 Configuration of each module in the network:** This task can be done after or before deploying the network. The client, the server and switch's controller are programmed. Necessary code for running the DTLS client and the one for running the DTLS server are programmed so as they will later be able to carry out a DTLS handshake. Furthermore, another script which runs on the controller is programmed somehow it handles the data (packets) received from the switch's control plane.

  Task duration: 50 hours

- **T3.3 Network deployment and application running:** This task consists of deploying the network and running the DTLS handshake. One must check that everything is working as expected.

  Task duration: 30 hours

11.2.4 WP4: Implementation of the entire network architecture

Having checked that the basic model works, now, an OCSP server and a CA are implemented at this point, EJBCA module. Thus, some extra configurations have to be added to the network.

- **T4.1 Network setup modification and deployment:** As a new entity has been added to the network model, some slight changes must be applied to general network configuration and to both switch data and control plane programs. Finally, the modified network is deployed.

  Task duration: 80 hours

- **T4.2 Configuration and deployment of EJBCA:** Once the network is deployed, EJBCA service needs to be set up. After applying required configuration, EJBCA service can be run.

  Task duration: 60 hours

- **T4.3 Application running:** With everything configured and deployed, the DTLS handshake must be run. When the final objective of the project is reached, one can move to the following work package.

  Task duration: 40 hours

11.2.5 WP5: Solution validation

Once having implemented and run the solution of the project, it must be checked that the system works properly according to its design and specifications. After checking that it works, several tests have been carried out so as to evaluate its performance.

. For that purpose, different timings are analysed in two different scenarios so as to compare time differences between them:

- **T5.1 Functional Validation:** In this task, Functional Validation Tests are performed. A first FVT is is made to check if the OCSP server responds correctly to OCSP requests. The next four tests are based on capturing traffic flow on different interfaces of the network model. Therefore, one can check if the system is exchanging desired data over the network.

  Task duration: 30 hours

- **T5.2 System Performance:** In order to perform this task, another scenario has been deployed so as to evaluate the system performance on the previous scenario and on the new one. The new scenario consists of a slight modification of the first one somehow validation functions, which previously were carried out by the switch's controller, are performed on the client side. As for the performance parameters, several timings are analysed in both scenarios to compare time differences between them and evaluate the effect that the proposed solution has on the network.

  Task duration: 40 hours

11.2.6 WP6: Project management and documentation

This work package is developed throughout the project, it involves all functions that manage and control how it progresses. This work package consists of two main tasks, which are described below:

- **T6.1 Monitoring and control of the project:** Project development is monitored through different meetings with the working team to assess project status, solve problems that may arise and, if so, to be able to propose alternative solutions. It is also necessary to check whether the project is roughly in line with the defined planification.

  Task duration: 600 hours

- **T6.2 Documentation:** The writing of the master thesis is a complementary task to the development of the project. All the work carried out regarding technical, economic, and social aspects during the realisation of the project is written up in a document. Furthermore, during this project another parallel document has been written noting each action carried out per day of working. This way, in the event of needing to start the project from start or if someone would like to use it as a basis for another project, a lot of time will be saved.

  Task duration: 600 hours

11.3 GANTT DIAGRAM

Table 13 shows Gantt diagram of the project

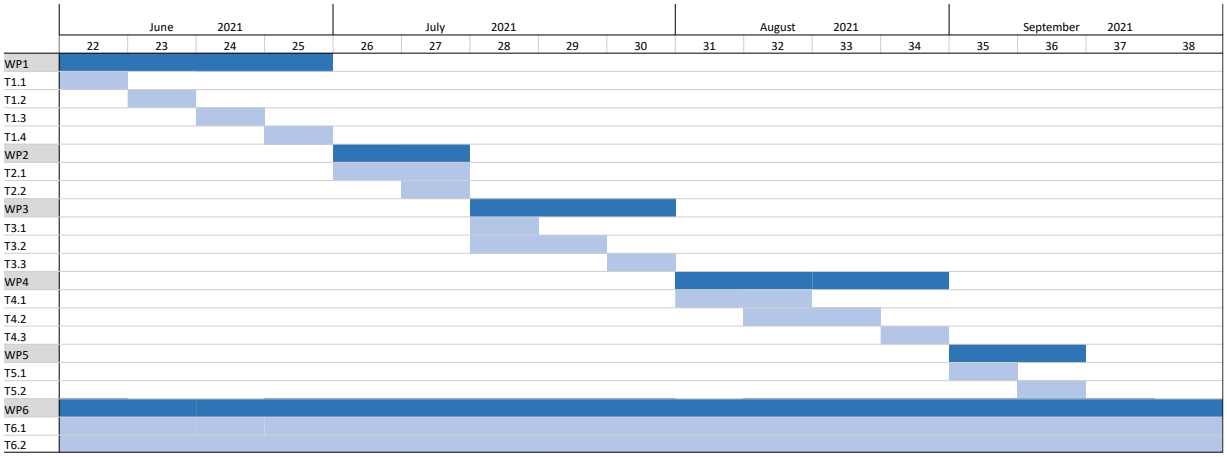

Table 13: Gannt diagram of the project

12 ECONOMIC COSTS

This section includes a summary of the project's budget. Development cost that the project implies will be explained.

12.1 HUMAN RESOURCES

Table 14 shows the project costs associated to human resources. One junior engineer and two Senior engineers are necessary. The junior is responsible for project's development whereas seniors oversee supervision tasks.

Concept	Quantity	Time (hours)	Cost (€/hour)	Total (€)
Junior Engineer	1	600	10	6.000
Senior Engineer	2	90	40	3.600
			Subtotal	**9.600**

Table 14: Costs of human resources

12.2 DEPRECIATION COSTS

This section describes the used resources for the development of the project. Table 15 shows depreciation costs.

Concept	Cost (€)	Usage (hours)	Lifetime (hours)	Total (€)
Microsoft Office License	80	250	1.500	13,33
HP Laptop	780	600	10.000	46,8
			Subtotal	**60,13**

Table 15: Depreciation costs

12.3 OTHER COSTS

Table 16 details the costs involved by other expenses.

Concept	Total (€)
Office Material	20
Laptop insurance	80
Internet connection	120
Subtotal	220

Table 16: Other costs

12.4 TOTAL COSTS

Finally, taking into consideration all three calculated costs in precious tables, total cost of the project has been calculated. Table 17 shows this total cost as well as subtotal. The latter has been calculated adding three previously calculated costs. Finally, a 10% for indirect costs is applied to the subtotal and a 10% for unforeseen costs is applied as well.

Concept	Total (€)
Human resources	9.600
Amortisations	60,13
Other costs	220
Subtotal	9.880,13
Indirect costs (10%)	988,01
Unforeseen costs (10%)	988,01
Total	11.856,15

Table 17: Total costs

13 CONCLUSIONS AND FUTURE WORK

After the development and completion of this project, it can be confirmed that the project satisfies the established requirements and has met the defined objectives. On the one side, it maintains DTLS communication standard, and all used software is Linux compatible. On the other side, it has been designed and implemented a secure, efficient, low-latency and robust system that has a programmable switch which carries out an IoT client's certificate validation tasks. In particular, verification of the signature and certificate revocation status checking.

Furthermore, the solution has been successfully validated. Then, it can be ensured that it works properly according to project specifications. Moreover, in order to prove that developed solution can be easily adapted to network requirements, the same solution has been implemented in a different environment with other network requirements, in this case the network demanded fragmentation on some packets of this project's DTLS handshake. The implementation on this environment ended in success as well. This solution has also been compared with a scenario where the client carries out all certificate validation functions, without any switch. Thus, a comparison between that handshake and the one performed in the implemented solution can be made.

The implemented solution does not interrupt the DTLS handshake, it adds nothing but slight time delay in the handshake in comparison with a handshake where the client carries out validation actions. Nevertheless, this delay is of the order of tens of milliseconds. Therefore, all the actions that happen from the point where the server sends the DTLS message with CA's and its certificate to the time when the client receives that message are transparent for both remote peers.

Thanks to the implemented system, as validations are done in the switch, it is ensured that these validations are being done on all clients. If certificate validation is left up to clients, they may or may not do it. Even though it is true that most DTLS implementations carry signature verification into effect, few of them implement OCSP query.

In addition, it has been demonstrated that thanks to In-Network Computing, actions that are developed in the switch's data plane are carried out much faster and more efficiently than if they are performed in the control plane, which uses higher level languages. Therefore, this solution tries to develop as many actions as possible at the switch's data plane so as to delegate the less actions to the controller and send the data to it in a way that simplifies the gathering of information at the controller.

The work with In-Network validation of certificates does not necessarily finish here. First of all, as already mentioned, the developed P4 program can easily be adapted to new demanded network features. This project has been adapted to two different scenarios regarding network requirements, however, there are still many more scenarios it can be adapted to. Moreover, this master's thesis has been developed for a specific server certificate, an interesting and future work could be developing a system that covers any IoT device of the wireless network. This could be done using variable headers that P4 offers, it requires a more fine-grained study of P4 programmable language. In addition, it would not be necessary to start it from scratch since this system can be used as an API to start to develop it.

14 REFERENCES

[1] What is IoT in simple words? *Azuko Technical Institute.* Retrieved September 19, 2021 from https://www.azukotech.com/post/what-is-iot-in-simple words#:~:text=The%20Internet%20of%20Things%20(IoT,wireless%20network%20without%20human%20intervention.

[2] Number of IoT connected devices worldwide 2019-2030. *Statista.* Published August 25, 2021 from https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/

[3] D.Cooper, S Santesson, S.Farrell, S.Boeyen, R. Housley, T.Polk. "Internet X.509 public Key Infrastructure certificate and certificate revocation list (CRL) profile". *RFC5280, IETF,* May 2008

[4] Dean Coclin. What is a CA? Certificate Authorities explained. Published May 26, 2021 from https://www.digicert.com/blog/what-is-a-certificate-authority

[5] E. Rescola, N. Modadugu. ¨Datagram Transport Layer Security Version 1.2¨ *RFC6347, IETF,* January 2012.

[6] Elena Georgescu. Heimdal security. ¨What is a Man'in the'Middle Attack?¨ Retrieved September 19, 2021 from https://heimdalsecurity.com/blog/man-in-the-middle-mitm-attack/

[7] Calum McClelland. What is an IoT Platform? Published January 10, 2020 from https://www.iotforall.com/what-is-an-iot-platform

[8] Govindraj Basatwar. ¨Guide to OWASP IoT Top 10 For Proactive Security¨. *AppSealing News.* Retrieved September 19, 2021 from https://www.appsealing.com/owasp-iot-top-10

[9] How Secure is my password? Retrieved 19 September, 2021 from https://howsecureismypassword.net/

[10] Dale Walker. ¨The top 12 password-cracking techniques used by hackers¨ Published September 7, 2021 from https://www.datacenters.com/news/top-password-hacking-methods-plus-10-tips-for-creating-strong-passwords

[11] Prateek Panda. ¨OWASP's Top 10 vulnerabilities and what you can do¨ Retrieved September 19, 2021 from https://www.intertrust.com/blog/owasps-top-10-iot-vulnerabilities-and-what-you-can-do/

[12] Wayne Thayer. ¨The Importance of Revocation Checking Part 2: A Real World Example¨ Posted March 11, 2021 from https://pkic.org/2013/03/11/the-importance-of-revocation-checking-part-2-a-real-world-example/

[13] Jake Frankenfield. ¨Eavesdropping attack¨ *Investopedia*. Updated October 2020 from https://www.investopedia.com/terms/e/eavesdropping-attack.asp

[14] Bill Holtz. ¨Certificate agility is just as important as crypto agility¨ Published July 9, 2021 from https://www.forbes.com/sites/forbestechcouncil/2021/07/09/certificate-agility-is-just-as-important-as-crypto-agility/?sh=47034ce96257

[15] Tom Warren. ¨Microsoft Teams foes down after Microsoft forgot to renew a certificate¨ Published February 3, 2020 from https://www.theverge.com/2020/2/3/21120248/microsoft-teams-down-outage-certificate-issue-status

[16] Lawrence Abrams. ¨Expired certificate led to an undercount of COVID-19 results¨ Published August 13, 2020 from https://www.bleepingcomputer.com/news/technology/expired-certificate-led-to-an-undercount-of-covid-19-results/

[17] Tim Callan. ¨Spotify outage: expired certificates continue to haunt systems¨ Retrieved September 19, 2021 on https://itsupplychain.com/spotify-outage-expired-certificates-continue-to-haunt-systems/

[18] Casey Crane. ¨New Study Finds 75% of CIOs are concerned about TLS Certificate-Related Security Risks¨ July 11, 2020 on https://securityboulevard.com/2020/07/new-study-finds-75-of-cios-are-concerned-about-tls-certificate-related-security-risks/

[19] S. Santesson, M.Myers, R.Ankney, A.Malpani, S.Galperin, C.Adams ¨X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP¨ *RFC6960, IETF*, June 2013

[20] Casey Crane. ¨Everything You Need to Know About OCSP, OCSP Stapling & OCSP Must-Staple¨ Published July 28, 2020 from https://www.thesslstore.com/blog/ocsp-ocsp-stapling-ocsp-must-staple/

[21] ¨How SDN based Architecture improve the security for IoT?¨ Retrieved 19 September, 2021 from https://www.bharatvikasnetworks.com/post/how-sdn-based-architecture-improve-the-security-for-iot

[22] ¨What is OpenFlow? Definition and How it relates to SDN¨ Published August 26, 2013 from https://www.sdxcentral.com/networking/sdn/definitions/what-is-openflow/

[23] ¨P4 Open Source Programming Language¨ Retrieved September 19, 2021 from https://p4.org/

[24] A.Almaini, A.Al-Dubai, I.romdhani, M.Schramm and A.Alsarhan, ¨Lightweight edge authentication for software defined networks¨, August 8, 2020

[25] E. Ollora, D.Franco, Z.Zhou and M.Berger, ¨P4Knocking: Offloading host-based firewall functionalities to the network¨ IEEE Xplore, April 9,2020

[26] R.Datta, S.Choi, A.Chowdhary and Y.Park, ¨P4Guard: Designing P4 Based Firewall¨, IEEE Xplore, January 3, 2019

[27] Z.Zuo, C.Chang and X.Zhu ¨A Software-Defined Networking Packet Forwarding Verification Mechanism Based on Programmable Data Plane¨, *Journal of electronics and information technology*, June 4, 2020

[28] A.C.Lapolli, J.A.Marques and L.P.Gaspary, ¨Offloading Real-time DDoS Attack Detection to Programmable Data Planes¨, *IEEE Xplore,* May 20, 2019

[29] F. Hauser, M. Häberle and M.Meth, ¨P4-IPsec: Site-to-Site and Host-to-Site VPN With IPsec in P4-Based SDN¨ IEEE Xplore, July 5, 2020

[30] Z.Zuo, R.He, X.Zhu and C.Chanh, ¨ A novel software-defined network packet security tunnel forwarding mechanism¨, *AIMS Press,* May 17, 2019

[31] P4 utils, Retrieved September 19, 2021 from https://github.com/nsg-ethz/p4-utils

[32] P4 Lang Tutorials, Retrieved September 19,2021 from https://github.com/p4lang/tutorials

[33] Python3 DTLS, Retrieved 19 September, 2021 from https://github.com/mobius-software-ltd/pyton3-dtls/tree/master/dtls/test

[34] Scandium, Californium, Retrieved September 19, 2021 from https://github.com/eclipse/californium.scandium

[35] XiPKI, eXtensible, sImple, Public Key Infrastructure, Retrieved 19 September, 2021 from https://github.com/xipki/xipki

[36] EJBCA, Enterprise Java Beans Certificate Authority, Retrieved 19 September, 2021 from https://doc.primekey.com/ejbca6152/tutorials-and-guides/quick-start-guide

[37] EJBCA, SourceForge repository, Retrieved 21 September 2021 from https://sourceforge.net/projects/ejbca/files/ejbca7/

ANNEXES

Annex I: Switch's data plane: P4 program

```
/* -*- P4_16 -*- */
#include <core.p4>
#include <v1model.p4>

const bit<16> TYPE_IPV4 = 0x0800;
const bit<8> PROTOCOL_UDP = 0x11;
const bit <8> CTYPE_HSHAKE = 0x16;
const bit<8> zeross = 0x00;

/*************************************************************************
********************** H E A D E R S ***********************************
*************************************************************************/
typedef bit<9> egressSpec_t;
typedef bit<48> macAddr_t;
typedef bit<32> ip4Addr_t;
typedef bit<8> contentType_t;

header ethernet_t {
    macAddr_t dstAddr;
    macAddr_t srcAddr;
    bit<16>   etherType;
}

header ipv4_t {
    bit<4>    version;
    bit<4>    ihl;
    bit<8>    diffserv;
    bit<16>   totalLen;
    bit<16>   identification;
    bit<3>    flags;
    bit<13>   fragOffset;
    bit<8>    ttl;
    bit<8>    protocol;
    bit<16>   hdrChecksum;
    ip4Addr_t srcAddr;
    ip4Addr_t dstAddr;
}

header udp_t {
    bit<16> srcPort;
    bit<16> dstPort;
    bit<16> length;
    bit<16> checksum;
}
```

```
header dtls_t {
      contentType_t contentType;
      bit<16> version;
      bit<16> epoch;
      bit<48> seqNumber;
      bit<16> length_;

}

header dtlsH_SH_t{
      bit<688> SH;
}

header dtlsH_preCert1_t{
      bit<144> certLength;
}

header dtlsH_Certificate1_t{
      bit<6960> certificate;
}

header dtlsH_preCert2_t{
      bit<24> certLength;
}

header dtlsH_Certificate2_t{
      bit<7024> certificate;
}

header dtlsH_postCerts_t{
      bit<104> certLength;
}

header dtlsH_SKE168_t{
      bit<2464>   ske;
}

header dtlsH_SHD_t{
      bit<96> shd;
}


struct metadata{
      /*empty*/
}




struct headers {
```

```
    ethernet_t               ethernet;
    ipv4_t                   ipv4;
    udp_t                    udp;
    dtls_t                   dtls;
    dtlsH_SH_t               dtlsHSH;
    dtlsH_preCert1_t         dtlsHPreCert1;
    dtlsH_preCert2_t         dtlsHPreCert2;
    dtlsH_Certificate1_t     dtlsHCertificate1;
    dtlsH_Certificate2_t     dtlsHCertificate2;
    dtlsH_postCerts_t        dtlsHPostCerts;
    dtlsH_SKE168_t           dtlsHSKE168;
    dtlsH_SHD_t              dtlsHSHD;

}


/*************************************************************************
********************** P A R S E R  ******************************
*************************************************************************/
parser MyParser(packet_in packet,
                out headers hdr,
                inout metadata meta,
                inout standard_metadata_t standard_metadata) {

    state start {
        transition select(standard_metadata.instance_type) {
            default: accept;
            }
      }

    state parse_ethernet {
        packet.extract(hdr.ethernet);
      transition select(hdr.ethernet.etherType) {
            TYPE_IPV4: parse_ipv4;
            default: accept;
            }
      }

    state parse_ipv4 {
        packet.extract(hdr.ipv4);
      transition select(standard_metadata.ingress_port) {
            0x01 :  parse_udp;
                  default : accept;
            }
      }




    state parse_udp {
```

```
    transition select(hdr.ipv4.protocol) {
        PROTOCOL_UDP : parse_dtls;
        default : accept;
        }
  }

state parse_dtls {
   packet.extract(hdr.udp);
   transition select(packet.lookahead<dtls_t>().contentType) {
              CTYPE_HSHAKE : parse_dtls_hshakeType;
              default : accept;
     }
}


 state parse_dtls_hshakeType {
      packet.extract(hdr.dtls);
      transition select(packet.lookahead<bit<8>>()){
            0x02 : parse_dtls_SH ;
        default : accept;
        }
  }

   state parse_dtls_SH{
      packet.extract(hdr.dtlsHSH);
      transition select(packet.lookahead<bit<8>>()){
        0x0B : parse_dtls_certificate;
        default : accept;
        }
  }

  state parse_dtls_certificate{
      packet.extract(hdr.dtlsHPreCert1);
      packet.extract(hdr.dtlsHCertificate1);
      packet.extract(hdr.dtlsHPreCert2);
        packet.extract(hdr.dtlsHCertificate2);
      packet.extract(hdr.dtlsHPostCerts);
      transition select(packet.lookahead<bit<8>>()){
        0x0C : parse_dtls_SKE168 ;
        default : accept;
        }
  }

  state parse_dtls_SKE168{
      packet.extract(hdr.dtlsHSKE168);
      transition select(packet.lookahead<bit<8>>()){
        0x0E : parse_dtls_SHD ;
        default : accept;
        }
  }

    state parse_dtls_SHD{
```

```
          packet.extract(hdr.dtlsHSHD);
          transition accept;
      }

}

/*************************************************************************
*********   C H E C K S U M    V E R I F I C A T I O N   *************
*************************************************************************/
control MyVerifyChecksum(inout headers hdr, inout metadata meta) {
    apply {  }
}

/*************************************************************************
**************  I N G R E S S   P R O C E S S I N G   ***************
*************************************************************************/
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t standard_metadata) {

    action drop() {
        mark_to_drop(standard_metadata);
    }

    action ipv4_forward(macAddr_t dstAddr, egressSpec_t port) {
        standard_metadata.egress_spec = port;
        hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
        hdr.ethernet.dstAddr = dstAddr;
        hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
    }

    table ipv4_lpm {
        key = {
            hdr.ipv4.dstAddr: lpm;
        }
        actions = {
            ipv4_forward;
            drop;
            NoAction;
        }
        size = 1024;
        default_action = drop();
    }


    apply {
      /**************FIRST DATAGRAM WITH CERTIFICATES**********/
      if(hdr.ipv4.isValid() && hdr.dtlsHCertificate1.isValid() &&
standard_metadata.instance_type == 0 && standard_metadata.ingress_port
!= 3){
            clone3(CloneType.I2E,100,meta);
            hdr.ipv4.totalLen = 1776;
```

```
            standard_metadata.egress_spec= 3;
      }

      /***************CLONED PACKET*****************************/
      else if(hdr.ipv4.isValid() && standard_metadata.instance_type ==
1){
            standard_metadata.egress_spec= 3;
        }

      /**************ANY OTHER IP PACKET********************/
        else if(hdr.ipv4.isValid() && standard_metadata.instance_type
== 0){
                 ipv4_lpm.apply();
        }
    }
}

/*************************************************************
***************  E G R E S S   P R O C E S S I N G   **************
*************************************************************/
control MyEgress(inout headers hdr,
                 inout metadata meta,
                 inout standard_metadata_t standard_metadata) {

apply {

      if(hdr.dtlsHCertificate1.isValid() &&
standard_metadata.instance_type == 0 && standard_metadata.ingress_port
!= 3){
            hdr.udp.length = 1756;
            hdr.dtls.setInvalid();
            hdr.dtlsHSH.setInvalid();
            hdr.dtlsHSKE168.setInvalid();
            hdr.dtlsHPreCert1.setInvalid();
            hdr.dtlsHPreCert2.setInvalid();
            hdr.dtlsHPostCerts.setInvalid();
            hdr.dtlsHSKE168.setInvalid();
            hdr.dtlsHSHD.setInvalid();
        }
   }
 }



/*************************************************************
*************   C H E C K S U M   C O M P U T A T I O N  **********
*************************************************************/
control MyComputeChecksum(inout headers  hdr, inout metadata meta) {

  apply {

      update_checksum(
```

```
            hdr.ipv4.isValid(),
              { hdr.ipv4.version,
              hdr.ipv4.ihl,
                hdr.ipv4.diffserv,
                hdr.ipv4.totalLen,
                hdr.ipv4.identification,
                hdr.ipv4.flags,
                hdr.ipv4.fragOffset,
                hdr.ipv4.ttl,
                hdr.ipv4.protocol,
                hdr.ipv4.srcAddr,
                hdr.ipv4.dstAddr },
              hdr.ipv4.hdrChecksum,
              HashAlgorithm.csum16);

          update_checksum(
              hdr.dtlsHSKE168.isValid(),
                  {    hdr.ipv4.srcAddr,
                          hdr.ipv4.dstAddr,
                    zeross,
                    hdr.ipv4.protocol,
                    hdr.udp.length,
                        hdr.udp.srcPort,
                        hdr.udp.dstPort,
                    hdr.udp.length,
                    hdr.dtls,
                      hdr.dtlsHSH,
                    hdr.dtlsHSKE168,
                    hdr.dtlsHPreCert1,
                    hdr.dtlsHPreCert2,
                    hdr.dtlsHPostCerts,
                    hdr.dtlsHSKE168,
                    hdr.dtlsHSHD,
                    hdr.dtlsHCertificate1,
                    hdr.dtlsHCertificate2},
                      hdr.udp.checksum,
                      HashAlgorithm.csum16);
        }

}

/*************************************************************************
 ***********************  D E P A R S E R  ***************************
 *************************************************************************/
control MyDeparser(packet_out packet, in headers hdr) {

   apply {
              packet.emit(hdr.ethernet);
                packet.emit(hdr.ipv4);
                packet.emit(hdr.udp);
              packet.emit(hdr.dtls);
              packet.emit(hdr.dtlsHSH);
```

```
            packet.emit(hdr.dtlsHPreCert1);
            packet.emit(hdr.dtlsHCertificate1);
            packet.emit(hdr.dtlsHPreCert2);
            packet.emit(hdr.dtlsHCertificate2);
            packet.emit(hdr.dtlsHPostCerts);
            packet.emit(hdr.dtlsHSKE168);
            packet.emit(hdr.dtlsHSHD);
    }
}


/*************************************************************************
 ***********************  S W I T C H ********************************
 *************************************************************************/
V1Switch(
MyParser(),
MyVerifyChecksum(),
MyIngress(),
MyEgress(),
MyComputeChecksum(),
MyDeparser()
) main;
```

Annex II: Switch's control plane: Controller python script

```
#!/usr/bin/env python
import sys
import struct
import os
import ssl
import socket
import argparse
import base64
import requests

from urllib.parse import urljoin
from scapy.all import sniff, sendp, hexdump, get_if_list,
get_if_hwaddr
from scapy.all import Packet
from scapy.all import UDP
from cryptography import x509
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.primitives.serialization import
load_pem_public_key
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.hazmat.primitives.hashes import SHA256
from cryptography.x509 import ocsp


def get_if():
    ifs=get_if_list()
    iface=None
    for i in get_if_list():
        if "eth0" in i:
            iface=i
            break;
    if not iface:
        print ("Cannot find eth0 interface")
        exit(1)
    return iface

#************************OCSP REQUEST**************************
def get_oscp_request(ocsp_server, cert, issuer_cert):
    builder = ocsp.OCSPRequestBuilder()
    builder = builder.add_certificate(cert, issuer_cert, SHA256())
    req = builder.build()
    req_path =
base64.b64encode(req.public_bytes(serialization.Encoding.DER))
    return urljoin(ocsp_server + '/', req_path.decode('ascii'))


#************************OCSP RESPONSE*************************
def get_ocsp_cert_status(ocsp_server, cert, issuer_cert):
```

```
    ocsp_resp = requests.get(get_oscp_request(ocsp_server, cert,
issuer_cert))
    if ocsp_resp.ok:
        ocsp_decoded = ocsp.load_der_ocsp_response(ocsp_resp.content)
        if ocsp_decoded.response_status ==
ocsp.OCSPResponseStatus.SUCCESSFUL:
            return ocsp_decoded.certificate_status
        else:
            raise Exception(f'decoding ocsp response failed:
{ocsp_decoded.response_status}')
    raise Exception(f'fetching ocsp cert status failed with response
status: {ocsp_resp.status_code}')

#************************EXTRACT CERTIFICATES**********************
def extract_certificates(payld):
      payld1=payld[:870]
      payld2=payld[870:1748]
      print(type(payld2), "\n", len(payld2))
      cert = x509.load_der_x509_certificate(payld1)
      issuer = x509.load_der_x509_certificate(payld2)
      return cert, issuer

#************************SIGNATURE*********************************
def verify_signature(issuer, cert):
      issuer_public_key = issuer.public_key()
      very = issuer_public_key.verify(
      cert.signature,
      cert.tbs_certificate_bytes,
      padding.PKCS1v15(),
      cert.signature_hash_algorithm,
      )
      return very

packets =[]
def handle_pkt(pkt):
      dport = pkt[UDP].dport

      if UDP in pkt and pkt[UDP].len == 2222:
            dport = pkt[UDP].dport
            pkt2 = pkt
            packets.append(pkt2)

      if UDP in pkt and pkt[UDP].len == 1756:
            payld = bytes(pkt[UDP].payload)
            issuer, cert = extract_certificates(payld)
            very = verify_signature(issuer, cert)
            if very == None:
                  very=0

      ocsp_server='http://10.0.4.4:8080/ejbca/publicweb/status/ocsp'
            url = get_oscp_request(ocsp_server, cert, issuer)
            status = get_ocsp_cert_status(ocsp_server, cert, issuer)
```

```
            if status.value == 0 & very == 0:
                  iface = get_if()
                  leng = len(packets)
                  pkt3 = packets[leng-1]
                  pkt3[UDP].dport = dport
                  sendp(pkt3, iface=iface)


def main():
    iface = get_if()
    print ("sniffing on %s" % iface)
    sys.stdout.flush()
    sniff(iface = iface, prn = lambda x: handle_pkt(x))


if __name__ == '__main__':
    main()
```

Annex III: Switch's configuration files:

Forwarding rules


```
{
  "target": "bmv2",
  "p4info": "build/basic.p4.p4info.txt",
  "bmv2_json": "build/basic.json",
  "table_entries": [
    {
      "table": "MyIngress.ipv4_lpm",
      "default_action": true,
      "action_name": "MyIngress.drop",
      "action_params": { }
    },
    {
      "table": "MyIngress.ipv4_lpm",
      "match": {
        "hdr.ipv4.dstAddr": ["10.0.1.1", 32]
      },
      "action_name": "MyIngress.ipv4_forward",
      "action_params": {
        "dstAddr": "08:00:00:00:01:11",
        "port": 1
      }
    },
    {
      "table": "MyIngress.ipv4_lpm",
      "match": {
        "hdr.ipv4.dstAddr": ["10.0.4.4", 32]
      },
      "action_name": "MyIngress.ipv4_forward",
      "action_params": {
        "dstAddr": "00:11:00:22:00:33",
        "port": 4
      }
    },
    {
      "table": "MyIngress.ipv4_lpm",
      "match": {
        "hdr.ipv4.dstAddr": ["10.0.2.2", 32]
      },
      "action_name": "MyIngress.ipv4_forward",
      "action_params": {
        "dstAddr": "08:00:00:00:02:22",
        "port": 2
      }
    },
    {
      "table": "MyIngress.ipv4_lpm",
      "match": {
        "hdr.ipv4.dstAddr": ["10.0.3.3", 32]
```

```
      },
      "action_name": "MyIngress.ipv4_forward",
      "action_params": {
        "dstAddr": "08:00:00:00:03:33",
        "port": 3
      }
    }
  ]
}
```

Clonning rule

```
mirroring_add 100 3
```

Annex IV: EJBCA setup configuration file:

Only configurable section is shown:

```
# Configurables
httpsserver_hostname="localhost"
database_host="10.0.4.4"
database_name="ejbcatest"
database_driver="org.mariadb.jdbc.Driver"
database_url="jdbc:mysql://${database_host}:3306/${database_name}?char
acterEncoding=UTF-8"
database_username="ejbca"
database_password="ejbca"
BASE_DN="O=Example CA,C=SE"
```