



UNIVERSITY OF LIÈGE - SCHOOL OF ENGINEERING AND  
COMPUTER SCIENCE

---

# OpenRoaming:

## Evaluation of the potential of e-ID as an Identity Provider in the OpenRoaming federation and implementation of a prototype

---

Academic supervisor :

Prof. Benoit DONNET

Cisco supervisors :

Mr. Bart BRINCKMAN

Mr. Hugues DE PRA

Mr. Frank DE REYMAEKER

Master's thesis completed in order to obtain the degree of Master of Science  
in Computer Science and Engineering, professional focus in computer  
systems and networks by

**Marie Maes**

ACADEMIC YEAR 2024-2025

# Acknowledgements

*First, I would like to thank my promoter, Professor Benoit Donnet, for his help in finding this internship, his support throughout the writing of this thesis, and for introducing me to the fundamentals of security, which is a subject that I am now particularly passionate about, through his course of Introduction to Computer Security.*

*A heartfelt thank you to Bart Brinckman for his huge help throughout this project. His thoughtful advice, regular calls, and constant guidance were essential. This thesis would not have been possible without him.*

*I am also very grateful to Hugues De Pra for introducing me to the world of Cisco and making me feel so welcome there.*

*Thank you to Frank De Reymaeker for his weekly check-in comments, which helped me stay on track.*

*Thanks to the Cisco team, especially Sebastien Marchal, Raphael Lienard and Guilian Deflandre for always letting me know they're available if I need them.*

*I would also like to thank Frédéric Pampalone for sharing with me his passion for IT and computer security. I am also deeply thankful to him and the CHC team for introducing me to my first tech-oriented student job, which was a really important experience in my journey.*

*I want to thank Florian Dekinder for his big support and its advices about the thesis writing process.*

*I also wish to express my gratitude to my parents. Thank you for your unconditional support during this TFE and these five years. Without you, nothing would have been possible. You are the best parents in the world.*

*Lastly, a big thank you to my family: my two big brothers and my big sister, for always being there and supporting me through everything.*

*Finally, I would like to acknowledge the use of generative AI tools in the writing process of this document. Specifically, Bridge i.t. [62], the Cisco AI assistant that answers questions relating to Cisco-internal content and built on OpenAI's ChatGPT, was used to help with drafting, bibliography, and editing content, and DeepL Write [31] was used for translation, writing and language support.*

# *Abstract*

---

UNIVERSITY OF LIÈGE - School of Engineering and Computer Science  
**OpenRoaming: Evaluation of the potential of e-ID as an Identity Provider  
in the OpenRoaming federation and implementation of a prototype**

Marie MAES

Supervisor: Prof. Benoit DONNET

Co-Supervisors: Mr. Bart BRINCKMAN, Mr. Hugues DE PRA, and Mr. Franck DE REYMAEKER  
Academic Year 2024-2025

---

In an era of growing need for network connectivity, traditional public Wi-Fi infrastructures face major limitation as they are either insecure or inconvenient if they require manual logins. To address these security and accessibility challenges, many Wi-Fi networks are now integrating with Identity Providers (IDP) and Access Network Providers (ANP). The IDP securely manages user identities and credentials, enabling more reliable and secure Wi-Fi access using user authentication, while the ANP manages network resources. OpenRoaming is a federation that enables easy Wi-Fi access across IDPs and ANPs.

The goal of this project is to evaluate how e-ID, the Belgian electronic identity card, can become an IDP in the OpenRoaming federation so that citizens can get seamless and secure Wi-Fi access using their e-ID credentials. This integration enables citizens who authenticate with their e-ID credentials via a mobile application to gain secure Wi-Fi access in government buildings and private venues without any manual configuration or interaction with their phone's Wi-Fi settings.

The project consists of three phases: (1) a theoretical study of OpenRoaming, e-ID, and related technologies, (2) the evaluation of potential approaches to integrate e-ID as an IDP, and finally (3) the development of a prototype. The components involved in this prototype include (a) a mobile application for the user to authenticate with e-ID, (b) an access point for managing Wi-Fi connections and forwarding authentication requests from the users, (c) a AAA server that includes an EAP/RADIUS server to communicate with the access point and a back-end server that will communicate with the IDP, and finally, (d) the IDP.

The final prototype demonstrates a secure and user-friendly system in which an Android device, after successfully being authenticated via the mobile application, seamlessly connects to previously unknown Wi-Fi networks in a safe environment. This is achieved through a robust configuration involving WPA2 Enterprise, EAP-TTLS with PAP over a RADSEC tunnel, OpenID Connect, and the use of certificates across all components.

This project successfully highlights how e-ID can become a reliable IDP in the OpenRoaming federation, addressing modern connectivity challenges while ensuring a secure user experience.

**Keywords:** *OpenRoaming; e-ID, Identity Provider, Wi-Fi.*

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Context . . . . .	1
1.2	Goal of this project . . . . .	3
1.3	Implementation of the prototype: Resources . . . . .	4
1.4	Roadmap . . . . .	4
<b>2</b>	<b>Theoretical Background: OpenRoaming</b>	<b>6</b>
2.1	OpenRoaming . . . . .	6
2.2	Wi-Fi and IEEE 802.11 Standards . . . . .	7
2.3	Wi-Fi Network Components and Management . . . . .	8
2.3.1	Wireless local-area network (WLAN) . . . . .	8
2.3.2	Access Point (AP) . . . . .	8
2.3.3	Service Set Identifier (SSID) . . . . .	9
2.3.4	Beacon Frames . . . . .	9
2.3.5	Wi-Fi Protected Access (WPA) . . . . .	10
2.4	OpenRoaming federation . . . . .	10
2.4.1	Access network Provider (ANP) . . . . .	11
2.4.2	Identity Provider (IDP) . . . . .	12
2.5	AAA framework, RADIUS and RADSEC . . . . .	13
2.5.1	AAA framework . . . . .	13
2.5.2	RADIUS . . . . .	14
2.5.3	RADSEC (RADIUS over TLS and TCP) . . . . .	15
2.6	TLS . . . . .	16
2.7	IEEE 802.1X and EAP protocol . . . . .	17
2.7.1	User authentication . . . . .	19
2.7.2	LEAP . . . . .	22
2.7.3	PEAP . . . . .	22
2.7.4	EAP-FAST . . . . .	24
2.7.5	EAP-TLS . . . . .	26
2.7.6	EAP-TTLS . . . . .	27
2.7.7	EAP-PPT . . . . .	29
2.8	Wireless Broadband Alliance (WBA) . . . . .	31

2.8.1	WBA-based Public key infrastructure (PKI)	31
2.9	Passpoint	33
2.9.1	Access Network Query Protocol (ANQP)	34
2.9.2	Roaming Consortium Organization Identifier (RCOI)	34
<b>3</b>	<b>Theoretical Background: e-ID</b>	<b>37</b>
3.1	Electronic identification (e-ID)	37
3.1.1	Multi-factor Authentication	38
3.2	OAuth 2.0 protocol and OpenID Connect (OIDC)	39
3.2.1	OAuth 2.0 protocol	39
41section*.18		
3.2.2	OpenID Connect (OIDC)	42
3.2.3	Tokens format	42
<b>4</b>	<b>Solution investigation</b>	<b>44</b>
4.1	Context for this project	44
4.2	Device (User's phone)	44
4.2.1	Programming language selection	45
4.2.2	Passpoint profile	45
4.3	Access Point	47
4.3.1	Meraki	47
4.3.2	OpenRoaming considerations: Cisco Spaces	48
4.4	RADIUS Server	48
4.4.1	Technologies used	48
4.4.2	EAP method selection	49
4.5	Auth server	50
4.5.1	Programming language selection	50
4.5.2	Database selection	51
4.6	IDP	54
4.6.1	Authentication method	54
4.6.2	Google credentials: Firebase/Google Identity	54
4.6.3	e-ID: FOD BOSA's FAS	57
4.6.4	IDP on-boarding	60
4.7	Final solution	61
<b>5</b>	<b>Prototype: implementation and demonstration</b>	<b>63</b>
5.1	Source code and other resources	63
5.2	Device (user's phone)	65
5.2.1	MainActivity	66
5.2.2	AuthScreen	66
5.2.3	AuthUtils	67
5.2.4	FMS	67

5.3	Access point . . . . .	68
5.4	OpenRoaming considerations: Cisco Spaces . . . . .	69
5.5	EAP/RADIUS server . . . . .	70
5.5.1	Certificates generation . . . . .	70
5.5.2	RADSEC configuration . . . . .	71
5.5.3	EAP module . . . . .	73
5.5.4	REST module . . . . .	74
5.6	Auth server . . . . .	75
5.6.1	Endpoints . . . . .	75
5.6.2	Token Handling . . . . .	78
5.6.3	External Service: Google OAuth Token Exchange . . . . .	79
5.6.4	Formats . . . . .	79
5.6.5	SSL configuration . . . . .	79
5.7	IDP (Firebase, Google) . . . . .	80
5.7.1	Firebase set up . . . . .	80
5.7.2	Google Identity set up . . . . .	81
5.8	IDP (BOSA, e-ID) . . . . .	82
5.9	Demonstration . . . . .	82
5.10	Use-cases . . . . .	90
<b>6</b>	<b>Conclusion</b>	<b>92</b>
6.1	Possible improvements . . . . .	93
	<b>Bibliography</b>	<b>95</b>

# List of Tables

2.1	IEEE 802.11 Wireless LAN based on [69] . . . . .	8
4.1	EAP types comparison . . . . .	49
4.2	Key-Value store database comparison based on [37] [29] . . . . .	53

# List of Figures

1.1	Users' belief in Public Wi-Fi from [107]	1
1.2	OpenRoaming federation	2
1.3	OpenRoaming network: eduroam on-boarding based on [102].	2
1.4	e-ID as an Identity Provider in the OpenRoaming federation	3
2.1	OpenRoaming federation: on-boarding flow taken from [8]	6
2.2	802.11 LAN architecture taken from [69]	9
2.3	Identity federation with many participants taken from [71]	11
2.4	IDP discovery call flow based on [71]	12
2.5	Workflow for RADIUS Authentication	15
2.6	TLS connection process based on [69]	16
2.7	802.1X Authentication Components	17
2.8	EAP authentication exchange based on [120]	19
2.9	PAP authentication based on [87]	20
2.10	CHAP authentication based on [87]	20
2.11	MS-CHAP authentication based on [87]	21
2.12	MS-CHAPv2 authentication based on [87]	21
2.13	PEAP message sequences for a successful Authentication via MS-CHAPv2 based on [94]	24
2.14	EAP-FAST message sequences for a successful Authentication based on [98]	25
2.15	EAP-TLS message sequences for a successful Authentication based on [104]	27
2.16	EAP-TTLS message sequences for a successful Authentication via Tunneled PAP based on [45]	29
2.17	EAP-PPT message sequences for a successful Authentication taken from [99]	30
2.18	Asymmetric Encryption process based on [36]	32
2.19	OpenRoaming federation architecture: WBA-based PKI taken from [8]	33
2.20	OpenRoaming RCOI format based on [136] [113]	35
2.21	IDP profile and ANP matching scenarios taken from [113]	36
3.1	e-ID-based PKI taken from [38]	37
3.2	OAuth 2.0 flow and the interaction between the four roles based on [57]	40
3.3	Tokens usage based on [57]	41
3.4	structure of a JWT based on [7]	43



4.1	OpenRoaming: Components of the project . . . . .	44
4.2	Home page: summary of the clients . . . . .	48
4.3	SSID overview with their configuration and computer Wi-Fi settings displaying them . . . . .	48
4.4	Meraki dashboard overview [20] . . . . .	48
4.5	Key-value store: concept . . . . .	52
4.6	The authorization sequence with Google APIs that use the OAuth 2.0 protocol, taken from [34] . . . . .	55
4.7	The authorization sequence with BOSA APIs that use the OAuth 2.0 protocol, taken from [44] . . . . .	58
4.8	OpenRoaming: Components of the project and associated technologies and softwares . . . . .	61
4.9	Final solution: EAP-TTLS with PAP over RADSEC taken from [27] and slightly modified to suit this project . . . . .	62
5.1	User interface for authentication with two buttons, one for Google and the other for e-ID . . . . .	65
5.2	Wifi suggestion pop-up to add the Passpoint profile . . . . .	65
5.3	Screenshots of the mobile application . . . . .	65
5.4	SSID Test1 configuration page for access control . . . . .	68
5.5	SSID Test1 configuration page for access control: RADIUS server configuration . . . . .	68
5.6	Meraki dashboard overview [20] . . . . .	68
5.7	The TLS Tunnel establishment, taken from [18] . . . . .	68
5.8	The TLS Tunnel establishment between the Meraki AP and the RADIUS server . . . . .	69
5.9	OpenRoaming Setup of the Test1 SSID in Cisco Spaces . . . . .	69
5.10	The Passpoint profile based on the structure described in [6] . . . . .	78
5.11	The SHA-1 of the signing certificate using the Gradle signingReport command in Android Studio . . . . .	80
5.12	Web client OAuth 2.0 ID from API Console, taken from [5] . . . . .	82
5.13	Welcome page of the mobile application that allows the user to authenticate . . . . .	83
5.14	Google screen to authenticate via the FirebaseAuth application . . . . .	83
5.15	Screenshots of the mobile application . . . . .	83
5.16	Logs of the auth server that show the requests to the auth and to the generateAndroidProfile endpoint . . . . .	84
5.17	Pop-up notification to allow suggested Wi-Fi networks . . . . .	85
5.18	OpenRoaming components summary . . . . .	85
5.19	Probe response information for the OpenRoaming-enabled SSID Test1 . . . . .	86
5.20	Event logs from the meraki access point . . . . .	86
5.21	EAP Identity exchange in the Access-Request RADIUS message . . . . .	87
5.22	Access-Request packet from the RADIUS server . . . . .	87

5.23	The TLS Tunnel establishment between the Meraki AP and the RADIUS server, in WireShark . . . . .	87
5.24	The TLS Tunnel establishment between the Meraki AP and the RADIUS server, in the FreeRADIUS logs . . . . .	88
5.25	Second phase of the EAP-TTLS exchange: the user sends its credentials . . .	88
5.26	REST call by the RADIUS server to the auth server to check the user credentials	89
5.27	Second phase of the EAP-TTLS exchange: the user sends its credentials . . .	89
5.28	Access-Accept packet from the RADIUS server . . . . .	89
5.29	Event logs from the meraki access point . . . . .	90
5.30	Screenshot of the user being connected to an OpenRoaming Wi-Fi . . . . .	90

# Listings

4.1	Profile with a username/password credential (EAP-TTLS) taken from [6] [35]	45
5.1	FreeRadius configuration: RADSEC client . . . . .	71
5.2	FreeRadius configuration: RADSEC configuration . . . . .	72
5.3	FreeRadius configuration: EAP module/EAP-TTLS . . . . .	73
5.4	FreeRadius configuration: PAP and EAP configuration . . . . .	74
5.5	FreeRadius configuration: REST module . . . . .	74
5.6	FreeRadius configuration: REST configuration . . . . .	75

# List of Acronyms

**AAA** Authentication, Authorization, and Accounting

**ANP** Access Network Provider

**ANQP** Access Network Query Protocol

**AP** Access Point

**API** Application Programming Interface

**BSS** Basic Service Set

**CA** Certificate Authority

**CHAP** Challenge Handshake Authentication Protocol

**CAG** Closed Access Group

**DNS** Domain Name System

**EAP** Extensible Authentication Protocol

**FAST** Flexible Authentication via Secure Tunneling

**HTTP** Hypertext Transfer Protocol

**HTTPS** Hypertext Transfer Protocol Secure

**IDP** Identity Provider

**IP** Internet Protocol

**LAN** Local area network

**LEAP** Lightweight Extensible Authentication Protocol

**NAI** Network Access Identifier

**OIDC** OpenID Connect

**PAP** Password Authentication Protocol

**PEAP** Protected Extensible Authentication Protocol

**PKI** Public Key Infrastructure

**RCOI** Roaming Consortium Organisation Identifier

**REST** Representational State Transfer

**SSID** Service Set Identifier

**SSL** Secure Sockets Layer

**TLS** Transport Layer Security

**TTLS** Tunneled Transport Layer Security

**WBA** Wireless Broadband Alliance

**Wi-Fi** Wireless Fidelity

**WLAN** wireless local-area network

**WPA** Wi-Fi Protected Access

# Chapter 1

## Introduction

### 1.1 Context

The demand for network access is constantly growing these days [123]. When people go out, they either turn on their mobile data or try to find a public Wi-Fi (section 2.2). Typically, when connecting to these Wi-Fi, a pop-up page appears and people are asked for a lot of information before they actually get access to the network. An alternative is to ask for the Wi-Fi name (called SSID) and the password when people are in a hotel, venue, enterprise, hospital, or any other location that offers Wi-Fi.

Needless to say, this is neither convenient nor secure, as these wireless networks are vulnerable to SSID name spoofing [59], network sniffing, and many other security threats. As can be seen in Figure 1.1, the problem is that one-third of the people connect with a public Wi-Fi daily. Even worse, almost 40% of the respondents are not concerned about the security of these Wi-Fi and think they are somewhat safe. The need for a secure and easy solution is thus a big challenge in this era of smartphone predominance.

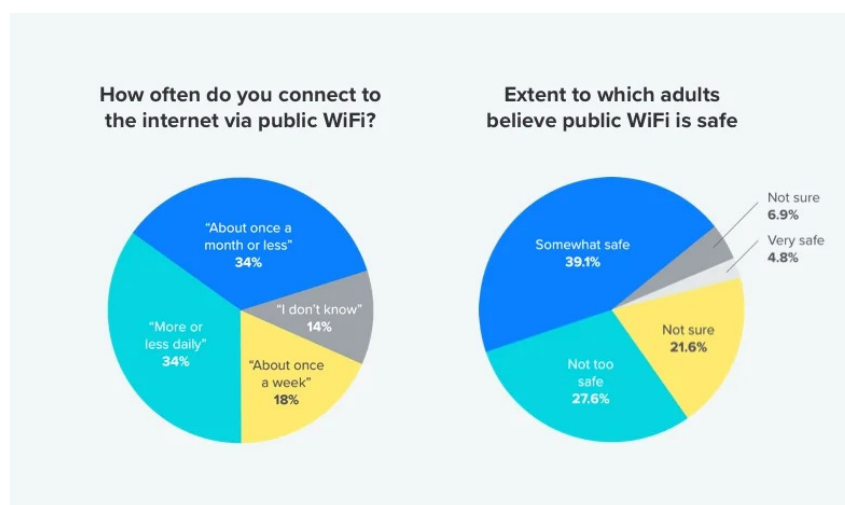


Figure 1.1: Users' belief in Public Wi-Fi from [107]

To address these security and accessibility challenges, many Wi-Fi networks are integrating with Identity Providers (IDPs) and Access Network Providers (ANPs). The

IDP securely manages user identities and credentials, enabling more reliable and secure Wi-Fi access using user authentication, while the ANP manages network resources and is the physical link between user's devices and the network. The relationship between IDPs and ANPs is critical, as it enables the establishment of federated networks, where users can authenticate across multiple networks seamlessly.

One of the solutions to manage this relationship between IDPs and ANPs is **OpenRoaming** (Section section 2.1), as represented in Figure 1.2. Developed and maintained by a global alliance called the Wireless Broadband Alliance (WBA) [135], **OpenRoaming** enables easy Wi-Fi access between identity providers (IDP) and access network providers (ANP). It allows users to connect to multiple Wi-Fi networks that are **OpenRoaming**-capable with a single initial connection. Once successfully authenticated, users do not need to re-enter their credentials when moving from one network to another.

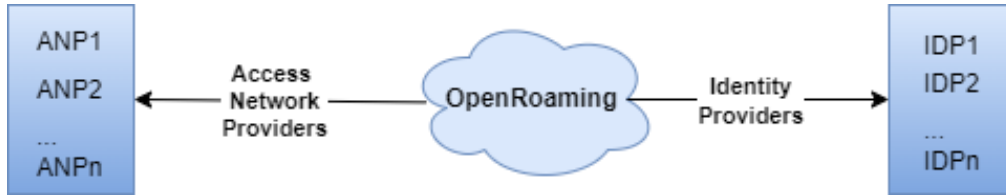


Figure 1.2: **OpenRoaming** federation

**OpenRoaming** is thus a framework that enables seamless and secure Wi-Fi access across different networks without requiring users to manually log in each time they change networks [135]. Its primary goal is to simplify the user experience by enabling automatic and secure Wi-Fi connections. This is particularly useful in environments such as airports, shopping centers and government buildings where users frequently move between different networks.

A well-known example of an **OpenRoaming**-enabled Wi-Fi network is the eduroam [127] [3] service, widely used in academic institutions, as shown in Figure 1.3. It is a wide

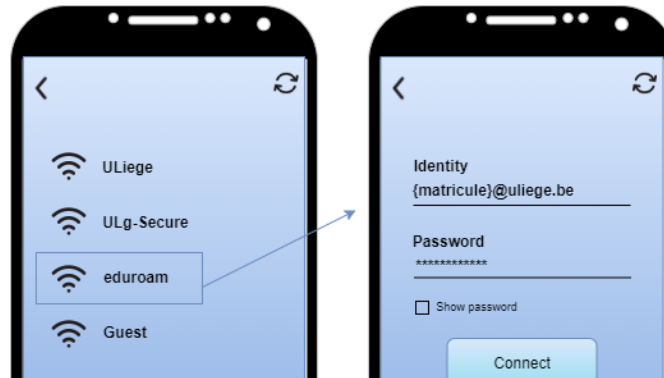


Figure 1.3: **OpenRoaming** network: eduroam on-boarding based on [102].

infrastructure including 28000 wired connection points and 1900 Wi-Fi access points [102]. Eduroam allows students, researchers, and staff to connect securely to the Wi-Fi network of any participating institution without needing to re-enter their credentials. For instance, a university student authenticated on their home campus can visit another university,

possibly in a different country, and automatically connect to the local eduroam network. This eliminates the need for manual logins, providing both convenience and security, even in unfamiliar locations.

## 1.2 Goal of this project

As mentioned above, **OpenRoaming** is based on an Identity federation with Access Network Providers (ANP) and Identity Providers (IDP) as members.

The goal of this project is to evaluate how **e-ID**, the Belgian electronic identity card, can become an IDP in the **OpenRoaming** federation, as can be seen in Figure 1.4, so that citizens can get seamless and secure **Wi-Fi** access using their **e-ID** credentials in government buildings or in private venues that want to offer connectivity to citizens.

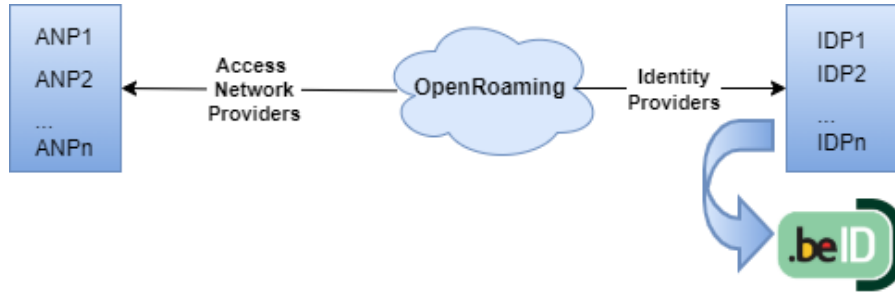


Figure 1.4: e-ID as an Identity Provider in the **OpenRoaming** federation

**e-ID** is a PKI-based solution that assigns authentication and signing certificates to citizens. The private keys of these certificates are securely stored on the chip card of the citizen's identity card. These certificates can be used to authenticate to government services or other services (e.g. banks) that accept the citizen's identity. The **e-ID** identity management solution supports OpenID Connect (OIDC) to authenticate to these web services. The **e-ID** can also be used to bootstrap 2-factor authentication using applications such as myGov [109] or Itsme [11].

The project consists of 3 phases:

1. The study phase: It consists of a theoretical study of **OpenRoaming**, **e-ID** and related technologies.
2. The solution investigation: This phase evaluates the potential approaches for integrating **e-ID** as an IDP, ranging from direct integration with EAP-TLS to a proxy between EAP and OIDC. It should also evaluate client-side approaches and possible integration with myGov application. Other aspects will also be evaluated, such as the choice of database and programming language.
3. The prototype: The final phase highlights the development of a demonstrable prototype of an **e-ID** IDP and a client that authenticates with **e-ID** credentials.



My personal contributions to this project include the 3 phases mentioned above, i.e. an in-depth study of the **OpenRoaming** technologies as well as the **e-ID** principles, then work towards a concept for a reliable solution to integrate **e-ID** as an IDP in the **OpenRoaming** federation, and finally implement this solution.

Through this work, several components are essential. It includes a mobile application for users to be able to authenticate with **e-ID**, an access point to manage the **Wi-Fi** connection and forward authentication requests, an **AAA** server that includes an **EAP/RADIUS** server and a back-end server to authenticate users and manage communication with an IDP, and finally the IDP.

The final prototype is a device that, after downloading the mobile application and successfully authenticating with the IDP, is able to seamlessly connect to a **Wi-Fi** that it had never visited before, while remaining in a safe environment.

### 1.3 Implementation of the prototype: Resources

The implementation of the prototype and various other resources is hosted in a gitlab directory:

<https://gitlab.uliege.be/Marie.Maes/openroaming>

A video demonstrating the whole solution is available at the following link:

OpenRoaming thesis - Demo and progress (Marie Maes)-20241219 1405-1.mp4

The solution presented in this video uses Google credentials instead of **e-ID** credentials. This is because integration with the **e-ID** IDP requires a special setup that should have been provided by the IDP providers. Unfortunately, it was not possible for them to do so in time for the delivery of this project. These credentials work almost identically, so switching from one authentication method to another is not a big adjustment.

### 1.4 Roadmap

The remainder of this report is divided into 4 chapters.

The chapter 2 discusses all the theoretical background required about **OpenRoaming** for this thesis. In particular, it focuses on **Wi-Fi**, **OpenRoaming**, identity providers (IDP) and related technologies.

The chapter 3 discusses the **e-ID** authentication method and related protocols. It focuses on **e-ID**, **OAuth 2.0**, **OIDC** and tokens.

The chapter 4 reviews the possible approaches available to implement the final solution, including the best **EAP** method for this use case and how to integrate **e-ID**.

The chapter 5 focuses on the implementation characteristics and peculiarities of the prototype.

Finally, a conclusion will be drawn about this work and its possible improvements in chapter 6.

# Chapter 2

## Theoretical Background: OpenRoaming

This chapter describes **OpenRoaming** and related technologies in more details to provide a foundation for the subsequent phases of this project.

### 2.1 OpenRoaming

The **OpenRoaming** [136] federation, as depicted in Figure 2.1, is composed of a few key components that interact with each other to enable secure and seamless **Wi-Fi** roaming. A reminder about **Wi-Fi** is given in section 2.2 and section 2.3.

**OpenRoaming** is developed and maintained by the WBA, an Alliance that will be described in section 2.8.

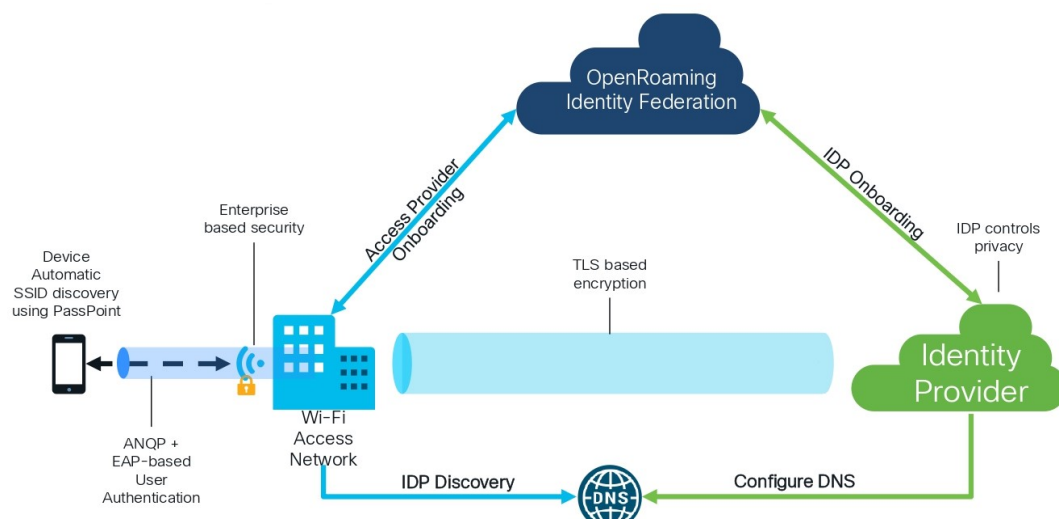


Figure 2.1: **OpenRoaming** federation: on-boarding flow taken from [8]

First, a device (for example, a smartphone) will automatically discover **Wi-Fi** networks that are PassPoint-enabled. The device uses the Access Network Query Protocol (ANQP) [112] to discover network details, including supported authentication methods and the identity of the network operator. The ANQP protocol is used in the Passpoint service

to help connect seamlessly. Passpoint technology and ANQP protocol are explained in section 2.9.

For this thesis, the authentication method will be **e-ID**, which is described in section 3.1. This authentication method supports OpenID Connect (**OIDC**), described in section 3.2.

Once the ANQP protocol discovers the network, it triggers the 802.1X [100] [70] authentication process to ensure that only authorized users can access the network. The 802.1X protocol relies on the Extensible Authentication Protocol (**EAP**) [120] to perform secure authentication between a supplicant (e.g., a user's smartphone) and an authentication server. The section 2.7 clarifies the 802.1x standard and the **EAP** protocol.

Note that there are several **EAP** types, but in the **OpenRoaming** case, **EAP-TTLS** [45] [8], which leverages **TLS**, is often selected. **TLS** [69] [95] is explained in section 2.6 and **EAP-TTLS** in section 2.7, along with other **EAP** types.

Figure 2.1 then represents the physical **Wi-Fi** infrastructure provided by the Access Provider (e.g., an airport, hotel, or ISP). This physical infrastructure includes the **Wi-Fi** Access Point (**AP**) to which users connect for internet access. The network is secured by enterprise security protocols, and more specifically the **WPA2** Enterprise [101] protocol. Typically, with **WPA2** Enterprise, an authentication server is used. It is usually a **RADIUS** server that is used to authenticate users and devices. The Access Providers and the **WPA2** protocol are described in section 2.3. **RADIUS** concepts are explained in section 2.5.

After that, to ensure that the user can connect seamlessly, the **Wi-Fi** Access Network needs to be part of the **OpenRoaming** federation as an Access Network Provider (**ANP**). Once it has joined the federation, it can participate in the **OpenRoaming** network, allowing its users to roam between different **Wi-Fi** networks easily.

The Federation also need on-boarded Identity Providers (**IDP**) to manage user authentication. Indeed, authentication requests are routed through the **OpenRoaming** federation to the appropriate **IDP**, which then verifies the user's credentials and grants access to the network. The details about the federation are presented in section 2.4 along with the concepts related to **ANP**'s and **IDP**'s.

## 2.2 Wi-Fi and IEEE 802.11 Standards

As **Wi-Fi** [125] and the IEEE 802.11 Standards [130] are the heart of this project, this section will briefly review these concepts [49] [70].

**Wi-Fi** is a wireless networking technology that allows devices to communicate over a wireless signal. It is based on the IEEE 802.11 standards that define the technical specifications for wireless local area networks (**WLANs**) and is widely used in local area networks (**LAN**). **Wi-Fi** operates on different channels that are at different frequencies, usually 2.4 and 5 GHz.

Table 2.1 shows the key versions of the **Wi-Fi** generations.

IEEE 802.11 standards	Year	Max data rate (Mb/s)	Range (m)	Frequency (GHz)
802.11b	1999	11	30	2.4
802.11a	1999	54	30	5
802.11g	2003	54	30	2.4
802.11n (Wi-Fi 4)	2009	600	70	2.4, 5
802.11ac (Wi-Fi 5)	2013	3470	70	5
802.11ax (Wi-Fi 6)	2021	14000	70	2.4, 5
802.11-2012	2012	54		2.4, 5
802.11af	2014	35-560	1000	unused TV bands (54-790 MHz)
802.11ah	2017	347	1000	900 MHz

Table 2.1: IEEE 802.11 Wireless LAN based on [69]

Also note that the IEEE 802.11u standard published in 2011 (which is included in 802.11-2012) will be used for SSID discovery and selection as it defines the Access Network Query Protocol (ANQP). ANQP will be discussed in subsection 2.9.1.

## 2.3 Wi-Fi Network Components and Management

This section dives into some of the important Wi-Fi Network Components [70] that will be used for this project.

### 2.3.1 Wireless local-area network (WLAN)

A Wireless local-area network (WLAN) [22] is a wireless network that allows wireless devices like smartphones, laptops, and other Wi-Fi-enabled devices to connect and communicate between each other and with wired infrastructures. A WLAN is a limited area like a home, office, or school and operates under Wi-Fi standards.

### 2.3.2 Access Point (AP)

Access Points (AP's) are essential components of Wi-Fi networks as they are the link between the wireless and the wired networks. It allows devices to connect to the wired infrastructure.

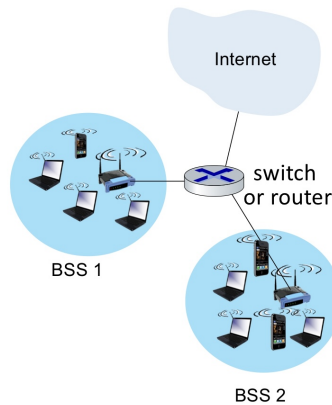


Figure 2.2: 802.11 LAN architecture taken from [69]

Figure 2.2 illustrate the 802.11 LAN architecture. The wireless host along with the AP form a Basic Service Set (BSS), also called a cell. These cells are connected from an AP to a router or a switch that then connects them to other cells and to the wired Internet.

AP's role is thus to allow wireless devices to connect to anything outside of their cell.

### 2.3.3 Service Set Identifier (SSID)

The Service Set Identifier (SSID) is a unique identifier assigned to a Wi-Fi network. It serves as a Wi-Fi network's "name" and is used to distinguish one wireless network from another. When a device (e.g., a smartphone) scans for available networks, it receives a list of SSID's from nearby access points. Users can then select the appropriate SSID to connect to the desired network. The screen on the left in Figure 1.3 shows a list of SSIDs, including eduroam, to which the user is connecting.

When a device arrives in a BSS, it needs to associate with an AP as this will be its link to the wired infrastructure. The AP's are used to broadcast a wireless signal containing an SSID so that devices can connect to it.

### 2.3.4 Beacon Frames

Beacons are a type of management frame in IEEE 802.11 networks, transmitted periodically by an AP to announce the presence of a wireless network. These beacon frames contain essential information about the network, including the SSID, supported data rates, and other network capabilities and details.

In the context of **OpenRoaming**, beacon frames play an additional role by advertising the Roaming Consortium Organization Identifier [136] [113] (RCOI), which signals that the AP is **OpenRoaming**-capable. RCOI's also allow to enforce different policies in the **OpenRoaming** federation. Depending on the RCOI, a user will be automatically rejected if he does not match the policy. The RCOI is provisioned via the **OpenRoaming** Passpoint profile and will thus be explained in more depth in the Passpoint section (section 2.9).

### 2.3.5 Wi-Fi Protected Access (WPA)

Wi-Fi Protected Access (WPA) [101] is a wireless security protocol developed by the Wi-Fi Alliance [125] to secure wireless networks. It provides robust encryption and authentication mechanisms to protect data transmitted over Wi-Fi networks, as these networks are unsecure channels.

The first version of WPA uses TKIP (Temporary Key Integrity Protocol) [132] as the encryption method. TKIP dynamically changes its key after they are used to encrypt. This ensures that a stolen key cannot be used for too long. WPA is now outdated as it has some limitations and vulnerabilities. Instead, WPA2 is used.

WPA2 [101][133] (IEEE 802.11i standard) provides stronger security by using a stronger encryption method than TKIP called AES [36][48] (Advanced Encryption Standard). AES uses a symmetric encryption algorithm [36] that is resistant to brute-force attacks. WPA2 supports two types of authentication modes:

- WPA2-Personal (WPA2-PSK): This mode uses a pre-shared key as a shared secret for authentication between the client and the access point. No AAA server is involved. When a user wants to get access to the network, he simply needs to enter the pre-configured password.
- WPA2-Enterprise: Typically, an AAA server is used (usually a RADIUS server). WPA2-Enterprise supports various EAP methods to provide secure user authentication and dynamic key generation. AAA servers are explained in section 2.5.

One of the key components of WPA2 is the Pairwise Transient Key (PTK). The PTK is derived during the WPA2 authentication process through a four-way handshake between the client and the access point using the EAPOL [86] [119] (Extensible Authentication Protocol over LAN) protocol. EAPOL is briefly explained in section 2.7. In WPA2-Enterprise, a Master Session Key (MSK) is generated during the EAP authentication phase between the client and the RADIUS server. The MSK is then used to derive the PTK and the PTK will be used with AES to encrypt data, ensuring secure data transmission.

## 2.4 OpenRoaming federation

The OpenRoaming federation is the core of the OpenRoaming architecture [136]. It is a federation of trusted IDP's and ANP's that enables seamless authentication and roaming between different Wi-Fi networks. IDP's and ANP's are combined under a same PKI-based trust architecture. The goal is to bring together a network of Wi-Fi networks [61], as shown in Figure 2.3. It also has a certificate authority and a revocation service running on openroaming.org.

The federation provides dynamic discovery based on DNS, and TLS-based [95] authentication and encryption, based on WBA-managed PKI which is described in subsection 2.8.1. Once an ANP has discovered a serving IDP for a user, and the ANP and IDP have mutually authenticated each other, the federation supports TLS-based services such as authentication,

accounting, settlement and metric exchange. In principle, any protocol can run across the federation, but in practice, the Extensible Authentication Protocol (EAP, explained in section 2.7) is used for network authentication. EAP can be leveraged to authenticate many credentials such as username/password, certificate, SIM card, token, etc. In this project, e-ID credentials will be used. e-ID will be explained in more details in section 3.1.

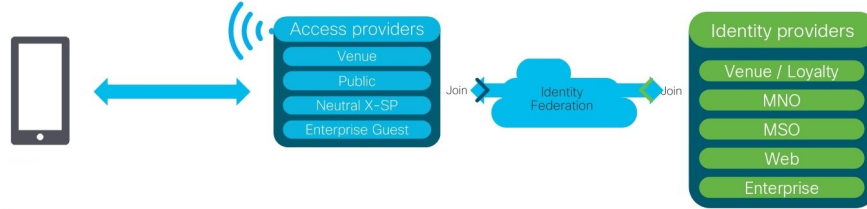


Figure 2.3: Identity federation with many participants taken from [71]

When a user connects to an **OpenRoaming**-enabled ANP, its device authenticates them with their IDP via the federation, which facilitates secure and trusted authentication without the user needing to log in manually.

TLS is used to protect the data in transit between the Access Network (ANP) and the Identity Provider (IDP). It ensures that sensitive information, such as authentication credentials, is encrypted during transmission. Before data exchange, the ANP must discover which IDP to use for a given user. The network uses DNS to find out which IDP is responsible for authenticating a user based on their domain or credentials. DNS also plays a role in routing authentication requests by resolving the domain name of the user IDP. This is part of the IDP discovery process.

### 2.4.1 Access network Provider (ANP)

An Access Network Provider (ANP) [136] in the **OpenRoaming** framework is an entity that operates a Wi-Fi network and provides users with internet access. To become an **OpenRoaming** ANP, an organization must apply to the Wireless Broadband Alliance (WBA) and agree to the terms of the **OpenRoaming** legal contract.

After receiving the certificate, the ANP configures its access points with the **OpenRoaming** Roaming Consortium Organization Identifier (RCOI). The RCOI is a global identifier advertised in beacons sent by AP's, indicating that the network is part of the **OpenRoaming** framework, and is described in more depth in subsection 2.9.2. This configuration allows mobile devices to easily discover and connect to the ANP's network.

ANP's then handles the on-boarding of mobile devices, which involves creating Wi-Fi profiles that enable devices to auto-connect to Wi-Fi networks. This can be done using the Passpoint mechanism (explained in section 2.9). When a device receives an **OpenRoaming** profile from an IDP, it can discover nearby **OpenRoaming** Wi-Fi networks through the RCOI. The device then starts a secure authentication process with the ANP using the information from the **OpenRoaming** profile.



### 2.4.2 Identity Provider (IDP)

An Identity Provider (IDP) [136] in the OpenRoaming framework is an entity that manages the identity and privacy of users. It authenticates the users with the given credentials and returns an authentication token upon successful authentication. It should also include OpenRoaming-related information called RCOI in the Passpoint profile of its users.

Any IDP should be on-boarded to become a member of the OpenRoaming federation [16]. Indeed, it needs to register with the OpenRoaming Domain Name System (DNS) to ensure that the associated realms are able to be discovered by the ANP's.

#### IDP discovery process [134]

Realms are part of the Network Access Identifier (NAI) [10]. NAI is a standardized identifier for users trying to get access to a network. The NAI structure consists of a username (or an anonymous identifier if the user wishes to remain anonymous) followed by an "@" symbol and a realm (e.g., username@realm.com).

In the OpenRoaming context, the NAI allows networks to identify where to forward authentication requests, ensuring that they reach the correct IDP. For example, the NAI *user123@exampleidp.com* is used as user identification (*user123*) and to route the authentication request to the right IDP (*exampleidp.com*).

In order to achieve this objective, NAPTR [72] and SRV [56] DNS records will be used. A NAPTR (Naming Authority Pointer) record is a type of DNS record that helps to dynamically discover services by specifying the preferred service, protocol, and additional lookup records. It defines rules that allow the system to find additional records (usually SRV records) needed to contact the right service. An SRV record is another type of DNS record that is used to specify the exact server and port where a specific service is hosted. SRV records identify the hostname (fully qualified domain name [129]) and port of the server that provides a particular service.

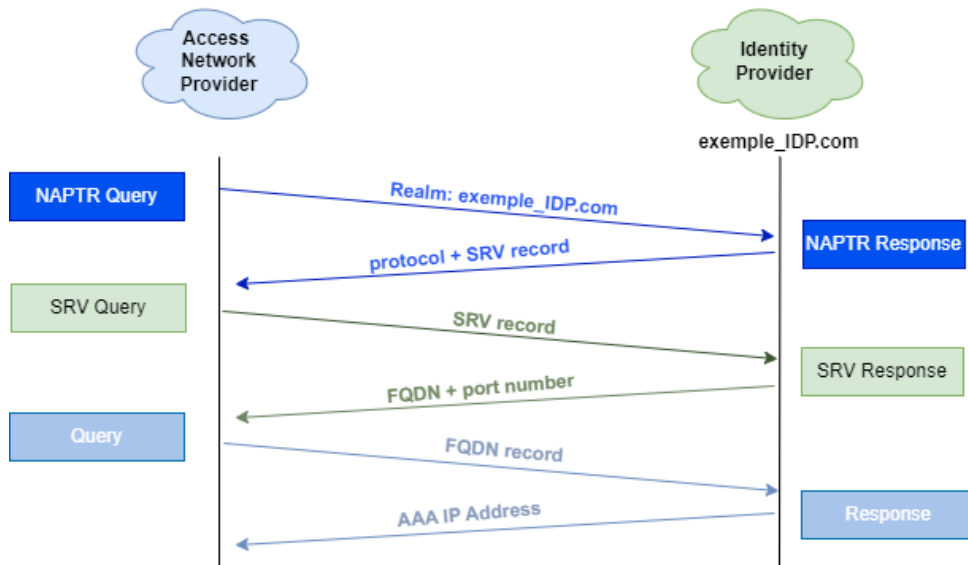


Figure 2.4: IDP discovery call flow based on [71]

As shown in Figure 2.4, 3 main queries are performed to discover the IDP:

1. **The NAPTR Query:** When the ANP receives an authentication request from a user (e.g., *user123@exampleIDP.com*), it extracts the realm (*exampleIDP.com*) from the user's NAI. The ANP sends an NAPTR query for the realm *exampleIDP.com* to discover which service and protocol to use when connecting to the IDP. The IDP's DNS server responds to the NAPTR query by providing a protocol and SRV record. The protocol tells the ANP how to communicate with the IDP's server.
2. **The SRV Query:** Using the protocol information, the ANP then sends an SRV query to find the exact server that provides the service for *exampleIDP.com*. The SRV query response returns the fully qualified domain name (FQDN) of the server along with the port number. This allows the ANP to locate the IDP authentication server precisely.
3. **The FQDN Query:** The ANP performs a DNS lookup for the FQDN received in the SRV response to get the IP address of the IDP server. The DNS server returns the IP address of the AAA server of the IDP, enabling the ANP to establish a connection with the IDP for the user's authentication request.

## 2.5 AAA framework, RADIUS and RADSEC

This section explains in more depth what is the AAA framework, RADIUS and RADSEC [111]. AAA (Authentication, Authorization, and Accounting) ensures secure access to network resources by verifying user identities, managing permissions, and tracking activity and metrics. RADIUS is a widely used protocol within this framework that relies on the unreliable UDP protocol. RADSEC improves RADIUS by leveraging TLS and TCP, providing encryption and reliable delivery, which improves security for modern networks.

### 2.5.1 AAA framework

AAA [13] stands for Authentication, Authorization, and Accounting and is a framework that helps manage users and their access to a network based on the user identity and the network policies implemented. It also helps keep track of user activity.

The first component of the framework is the authentication. It helps authenticating a user via credentials such as username and password to verify the identity of the user.

If the user is correctly authenticated, the second component of the framework, Authorization enters the scene. Its purpose is to grant some privileges to the user. These privileges include what the user is allowed to do and which resources and services he can access.

The last component is accounting. It is used to log the user's activity, for example, which resources were used and for how long. This tracking can be used for billing or building statistics.

### 2.5.2 RADIUS

Remote Authentication Dial-In User Service (RADIUS) [96] is an implementation of the AAA framework. RADIUS is a client-server protocol that is used to authenticate, authorize and account users.

There are 4 main types of RADIUS packets:

1. **Access-Request:** It is sent to the RADIUS server when a client initiates the authentication process and can contain information such as the username and password of the user.
2. **Access-Accept:** It is used when the RADIUS server grants access to the network to the user.
3. **Access-Reject:** It denies the user access to the network.
4. **Access-Challenge:** It is sent by the RADIUS server in response to an Access-Request if the request requires additional information from the user to accept it. Access-Challenge is often triggered by protocols such as EAP (Extensible Authentication Protocol) [45], which can require multiple message exchanges between the client and the RADIUS server for successful authentication [12]. EAP will be described in section 2.7.

As confidential information such as passwords can be transmitted via the RADIUS protocol, some mechanisms are implemented to secure the RADIUS communication, such as a secret. The RADIUS secret is a shared password that must be configured identically on both the RADIUS client and the RADIUS server and is never transmitted over the network. Instead, it is used to verify the authenticity of each device. When a RADIUS client sends an Access-Request to the RADIUS server, it includes an Authenticator field that is encrypted using the RADIUS secret. The RADIUS server then uses the shared secret to decrypt and verify the Authenticator, confirming that the request came from a trusted client. The RADIUS secret is used to encrypt sensitive parts of RADIUS packets, particularly the user's password in Access-Request packets.

Typically, users do not connect directly to the RADIUS server, but rather to an AP that is the link between the users and the RADIUS server. The AP is then responsible for routing authentication requests from the users to the server. A typical RADIUS flow is shown in Figure 2.5.

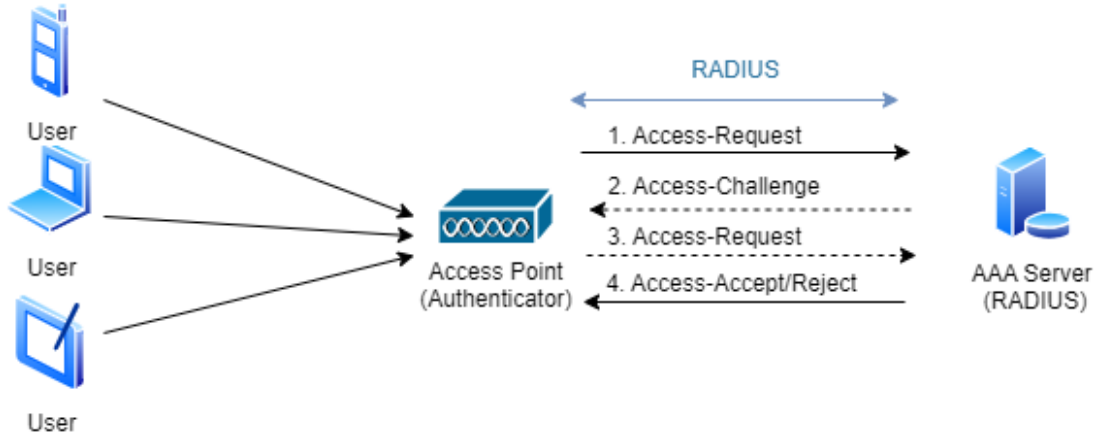


Figure 2.5: Workflow for RADIUS Authentication

The client initiates the connection through an access point that then sends an Access-Request packet with the user’s identity information (such as a username) to the RADIUS server. The RADIUS server can then determine that further information (like a password or encrypted credentials) is needed to complete the authentication and responds with an Access-Challenge packet. The client provides the requested information (e.g., password, certificate) in response to the challenge via a new Access-Request packet sent back to the RADIUS server. The RADIUS server then verifies the credentials from the second Access-Request and valid them or not. If valid, the AP grants network access to the user.

### 2.5.3 RADSEC (RADIUS over TLS and TCP)

A weakness of the RADIUS protocol is that it runs on the unreliable transport protocol UDP [1]. This lack of reliability can be problematic in network environments where security and reliability are critical, such as the authentication process in the RADIUS environment.

To address these issues, RADSEC (RADIUS over TLS and TCP) [126] [55] was introduced. RADSEC uses TLS (Transport Layer Security) [95] and TCP (Transmission Control Protocol) [2] to add encryption and reliable transport to RADIUS communications. Indeed, TLS provides strong encryption, ensuring data confidentiality and integrity, while TCP, unlike UDP, enhances reliability as it guarantees packet delivery and maintains packet order. A well-known service that uses RADSEC is the roaming environment eduroam [127] [3].

The default destination port number for RADSEC is 2083. RADSEC works in the same way as RADIUS, but first establishes a TCP connection. After the TCP handshake is done, a TLS session is negotiated. TLS is explained in more details in section 2.6. The server certificate is thus validated by the client assuming that the client is configured to trust the CA that signed the server certificate. This prevents wireless devices from connecting to a malicious server.

## 2.6 TLS

The TLS [69] [95] protocol, as can be seen on Figure 2.6, is a protocol on top of TCP that provides security and data integrity, usually between a client and a server.

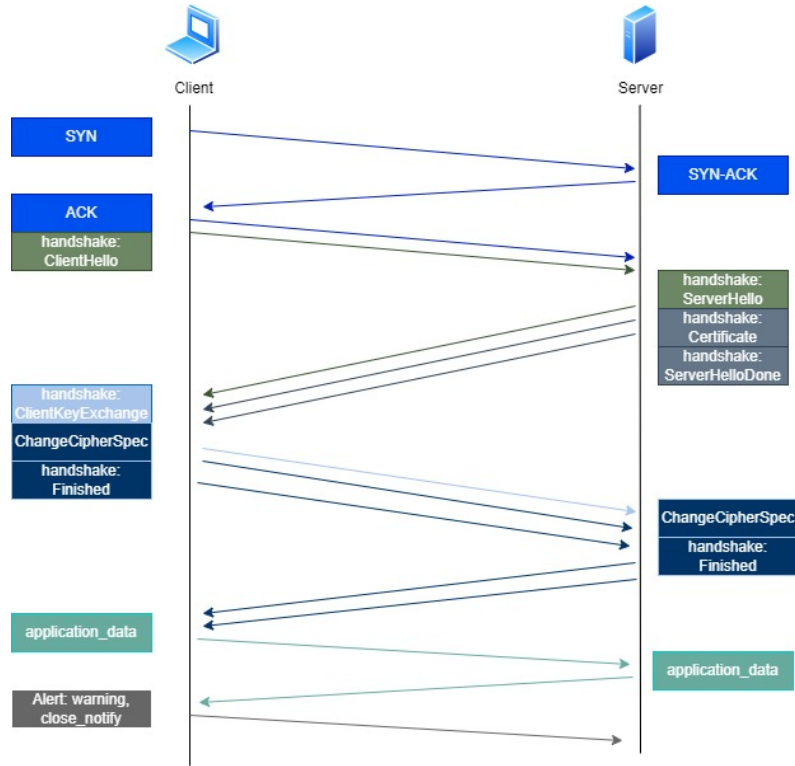


Figure 2.6: TLS connection process based on [69]

After a successful TCP connection between the client and the server using the three-way handshake (SYN, SYN-ACK, and ACK), there are four phases to the TLS Handshake Protocol [69]:

1. Establish Security Capabilities: the client and server negotiate the cryptographic parameters they will use during the session using the ClientHello and ServerHello messages. This will allow them to select a common cryptographic algorithm.
2. Server Authentication (and Key Exchange): The server sends its certificate to the client who will check its validity. In some cases, the server may also send a ServerKeyExchange message that contains additional information for the key exchange. Once the server has completed these steps, it sends a ServerHelloDone message to notify the client that it has finished.
3. (Client Authentication and) key exchange: If the server has requested client authentication (which is optional), the client may send its certificate to the server. The client then generates a pre-master secret (PMS), encrypted with the server's public key and sends it as a ClientKeyExchange message.

4. **Finish:** It starts with a ChangeCipherSpec message from both the client and the server that indicates that everything is now encrypted and that integrity is protected. Both of them then send a Finished message that is encrypted and that verifies that the key exchange and authentication processes were successful.

After the TLS handshake is completed successfully, the encrypted channel is ready for application data exchange. An Alert message (from the TLS Alert Protocol) can be sent to report errors or notify the closure of the TCP connection.

## 2.7 IEEE 802.1X and EAP protocol

802.1X [100] [70] is a set of standards specified for Wi-Fi, and works using a three-part framework, whose components are represented on Figure 2.7:

- **The supplicant:** The device (for example, a smartphone) that is trying to access the network and therefore must provide its credentials to the authenticator.
- **The authenticator:** The access point (AP) that requests the credentials of the supplicant and that manages communication between the supplicant and the authentication server.
- **The authentication Server:** An AAA server that stores and manages the credentials for users. Typically, it is a RADIUS server that is responsible for validating the user credentials and granting or denying access.

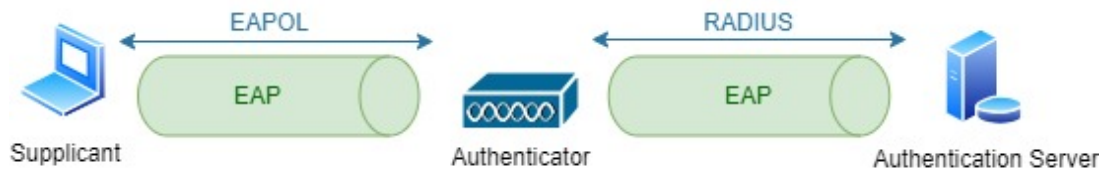


Figure 2.7: 802.1X Authentication Components

Between the suppliant and the authenticator, EAPOL [86] [119] is used, while between the authenticator and the authentication server, RADIUS is used. EAPOL is used in wired and wireless networks to facilitate communication between a supplicant and an authenticator as it encapsulates EAP messages and transports them over Ethernet frames (EAPOL-EAP packet). Other types of packets can also be used for initiating user authentication (EAPOL-Start packet), and for key management in WPA2 (EAPOL-Key packet).

The 802.1X protocol relies on the Extensible Authentication Protocol (EAP) [120] to perform secure authentication between a supplicant and an authentication server.

There are 4 types of EAP packets:

1. **Request:** Sent by the authenticator to the client to request some information
2. **Response:** Sent by the client to provide some information to the authenticator

3. **Success:** Sent by the authenticator to the client to notify the client that he is authenticated successfully
4. **Failure:** Sent by the authenticator to the client to notify the client that he cannot be authenticated

The EAP exchange is shown in Figure 2.8. The client begins the authentication process by sending an EAPOL-Start message to the access point. The access point responds to the client with an EAP-Request for Identity and the client responds with an EAP-Response/Identity, sending its identity to the access point. The access point forwards the client's EAP-Response/Identity to the RADIUS server by encapsulating it in a RADIUS Access-Request message. The access point essentially passes the EAP message through to the RADIUS server. The RADIUS server processes the identity and sends an EAP-Request in the form of a RADIUS Access-Challenge message back to the access point. This challenge starts an EAP authentication exchange. The client responds to the challenge with an EAP-Response, which may contain additional authentication information, such as a password or a response to a challenge. Depending on the type of EAP authentication method used, this challenge step might be repeated as many times as needed. Once the RADIUS server has verified the client's identity, it sends an EAP-Success message within a RADIUS Accept message to the access point and the access point forwards the EAP-Success message to the client, indicating that authentication has been successfully completed.

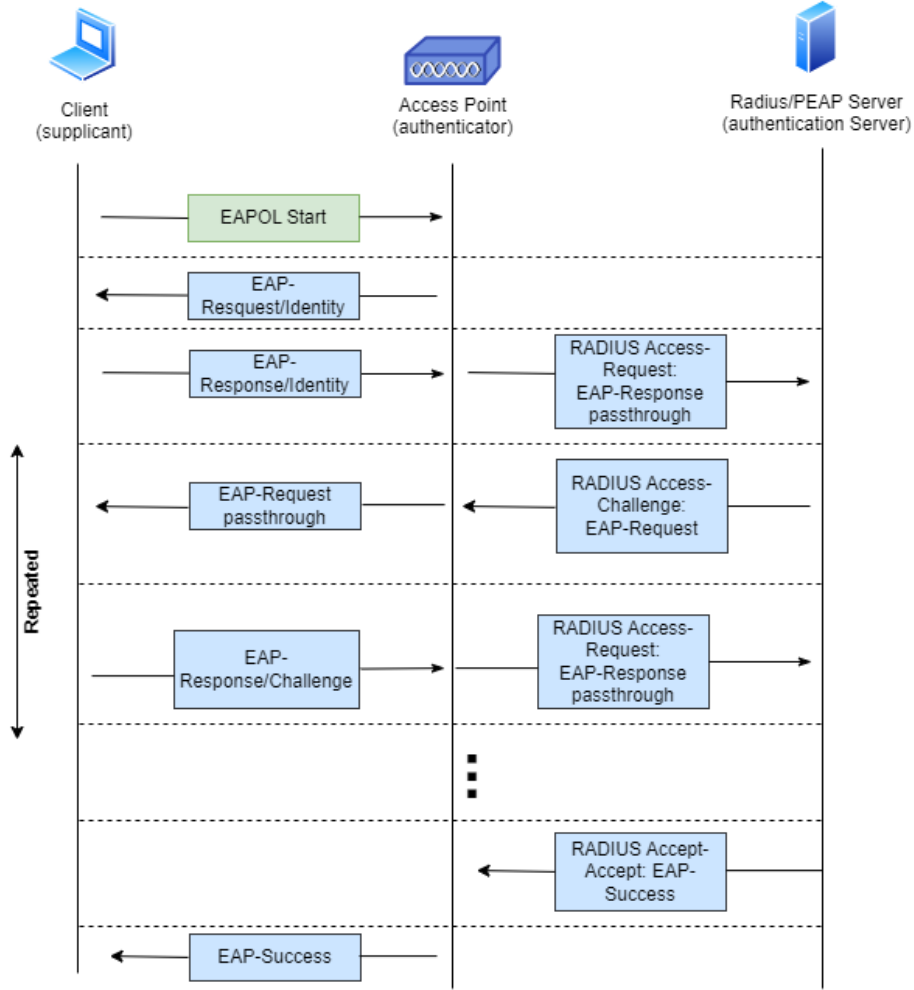


Figure 2.8: EAP authentication exchange based on [120]

EAP is able to carry a lot of different user authentication methods (known as EAP types), such as LEAP, EAP-FAST, PEAP, EAP-TLS, and EAP-TTLS. For the OpenRoaming setup, EAP-TTLS is often selected due to its balance between security, scalability and flexibility [137] [127]. For this project, the EAP method selected will be discussed in section 4.4.2. Let us first dive into the different methods and their characteristics.

### 2.7.1 User authentication

In many EAP methods, some inner user authentication protocols such as PAP, CHAP, and MS-CHAPv2 are commonly used to verify a client's identity. These protocols provide a way to securely authenticate the user by exchanging credentials within an encrypted channel, ensuring that only authorized users get access to the network. Originally developed for the use in PPP (Point-to-Point Protocol) [106] connections, which enable direct communication between two nodes, these authentication protocols are necessary to secure network authentication. Let us focus on them before explaining the different EAP methods.



## PAP

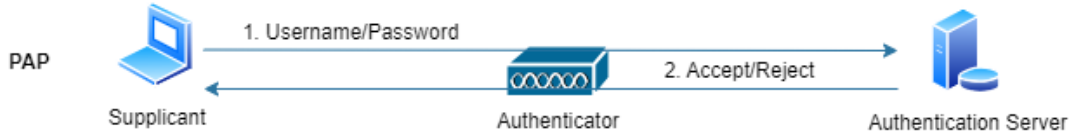


Figure 2.9: PAP authentication based on [87]

The Password Authentication Protocol (PAP) [105] is the simplest authentication protocol used to authenticate a supplicant (that is, the client) within the EAP exchange. The supplicant sends its username and password (referred to as an Id/Password pair) in plaintext to the authenticator, which then passes these credentials to the authentication server. This simple method allows the server to verify the provided credentials against its user database and either accepts or rejects the connection. This is thus a simple 2-way handshake. PAP by itself is not a strong authentication method as the password are sent in plaintext and its utilization is expected to be done within a secure environment, such as a TLS tunnel.

However, security mechanisms are implemented. In certain configurations, PAP can be adapted to improve security by using a one-way hash. Instead of sending the plaintext password, the supplicant hashes the password using a cryptographic hash function and transmits this hash as the credential. The authentication server stores the hashed version of each user's password in its database. When it receives the hashed password from the supplicant, it directly compares it with the stored hash value to authenticate the client. This approach ensures that the actual password is never transmitted, and only the hash is exchanged [87].

## CHAP

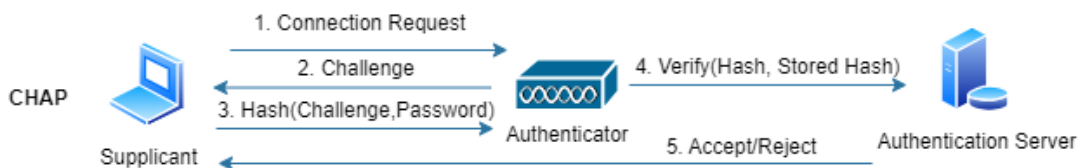


Figure 2.10: CHAP authentication based on [87]

Challenge-Handshake Authentication Protocol (CHAP) [105] is a protocol that is considered more secure than PAP because it does not transmit passwords in plaintext. CHAP uses a challenge-response process, which is often described as a three-way handshake to authenticate users.

When a client (supplicant) initiates a connection, the authenticator generates a unique challenge and sends it to the client. The client then combines the received challenge with its password to compute a hash value. This hash value is sent back to the authenticator.

Upon receiving the response, the authenticator retrieves the stored client password from its database and performs the same hashing operation using the received challenge and the stored password. If the hash values match, the client is authenticated successfully.

To perform this comparison, CHAP therefore requires the authenticator to retrieve the client's password from storage to compute the expected response. Because CHAP's challenge-response mechanism relies on combining the challenge with the actual client's password (or its hash), two-way encryption is necessary for the authenticator to get access to the stored password securely.

## MS-CHAP and MS-CHAP-V2

Microsoft Challenge-Handshake Authentication Protocol (MS-CHAP) [139] and its improved version, MS-CHAPv2 [138], are authentication protocols developed by Microsoft to address some of the security limitations of CHAP and are often used in environments that require compatibility with Windows systems (but are also supported on Linux and macOS).

### MS-CHAP

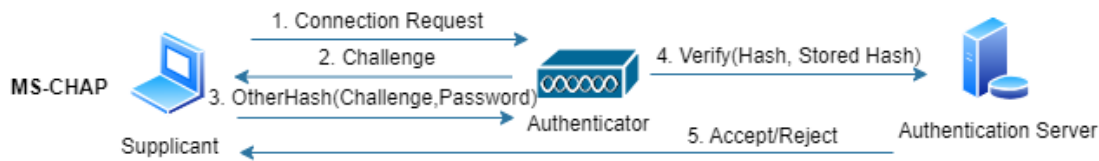


Figure 2.11: MS-CHAP authentication based on [87]

MS-CHAP is based on the basic CHAP challenge-response mechanism, except that it uses a variant of the MD4 hash instead of MD5, and the responses are encrypted with DES. MS-CHAP has known vulnerabilities [87] such as offline dictionary attacks, and lacks mutual authentication, meaning only the client is authenticated, not the server. This version is no longer used and MS-CHAPv2 is preferred.

### MS-CHAPv2

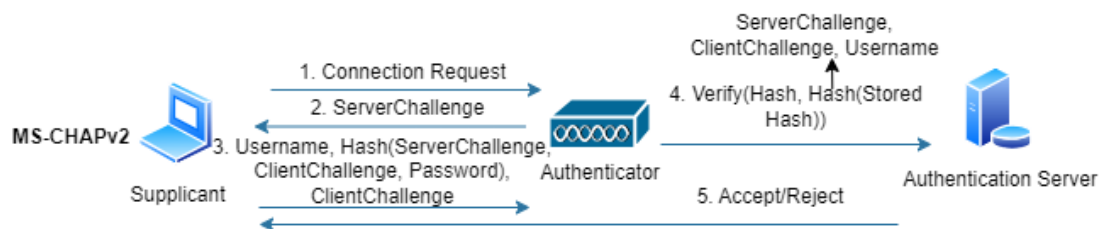


Figure 2.12: MS-CHAPv2 authentication based on [87]

MS-CHAPv2 is similar to CHAP, but introduces mutual authentication in addition. When the client receives the challenge from the server, it combines the server's challenge, the

client's challenge, and the hashed password to create a response. It then sends back the username, the client challenge, and the response to the server. The server retrieves the hashed password associated with the username from its database but does not directly compare it to the response from the client. Before, it hashes the database hash using a hash of the username, client challenge and server challenge via several cryptographic steps that are not detailed here. The server compares the result of its encryption with the response received from the client. If they match, the authentication is successful.

### 2.7.2 LEAP

LEAP (Lightweight Extensible Authentication Protocol) [121] [67] is a Wi-Fi authentication mechanism developed by Cisco. LEAP is considered lightweight as it does not use any certificate on the client or server side. It authenticates users through a username and password that must be validated before the user gets access to the network.

Concerning its security, LEAP uses mechanisms such as Wired Equivalent Privacy (WEP) [101] which uses dynamic keys that prevent an attacker from using a stolen key indefinitely. Despite this, WEP is the ancestor of WPA and WPA2 and is now considered obsolete as it presents huge security vulnerabilities. Indeed, LEAP is prone to dictionary attacks [36] which is a type of brute-force attack where the goal is to try all possible passwords until you get a match. As LEAP transmits cleartext usernames and encrypted passwords, an attacker can simply intercept them and perform a dictionary attack on the encrypted password until he gets a match.

Although obsolete, LEAP is sometimes used for its simplicity for applications that do not require security mechanisms at all.

### 2.7.3 PEAP

PEAP (Protected Extensible Authentication Protocol) [67] [91] [94] is a more secure alternative to LEAP as it uses an encrypted TLS tunnel and certificates on the server side. This protects the exchange against offline dictionary attacks.

PEAP establishes a secure TLS tunnel between the supplicant and the authentication server. It requires a server-side PKI certificate to create this tunnel so that public key of the server can be used for encryption.

Figure 2.13 shows the PEAP message sequence for successful authentication using MS-CHAPv2, as this is a common inner authentication for PEAP. Note that the sequence of messages remains the same if CHAP or MS-CHAPv1 were used as the specifics of the inner EAP method are encapsulated within the secure TLS tunnel and is represented by the EAP-Response/Identity and EAP-Response/Inner EAP messages. The format of the exchange is thus made generic as CHAP, MS-CHAPv1 and MS-CHAPv2 all use the same schema of a connection request (EAP-Response/Identity) from the client, a challenge and a response to this challenge (EAP-Response/Inner EAP) from the client.

As soon as the client has initiated the connection, EAP is started with the authenticator sending an EAP-Request/Identity packet to the client. The client responds with an EAP-Response/Identity message containing its identity. This identity is typically in the form of a username or a network access identifier (NAI). The NAI is used to identify the client and may also indicate the realm or domain to which the client belongs (e.g., *user123@exampleidp.com*). The realm can be used by the authentication server to route the authentication request to the appropriate domain. The authenticator forwards the EAP-Response/Identity to the server as part of a RADIUS Access-Request message. The EAP conversation in PEAP involves two phases:

- **Phase 1: Establishing a Secure Tunnel (TLS Handshake).** This phase establishes a secure encrypted TLS tunnel between the client (supplicant) and the authentication server (RADIUS server). During this phase, TLS is used to authenticate the PEAP server with the client, as already seen in section 2.6. EAP packets are exchanged between the client and the PEAP server through the access point (authenticator), which simply forwards these packets without modification. This exchange completes the TLS handshake, establishing a secure encrypted tunnel between the client and the PEAP server. Once the TLS handshake is complete, the remainder of the communication, including the inner authentication phase, occurs within this secure tunnel.
- **Phase 2: Inner Authentication (MS-CHAPv2).** After the TLS tunnel is established, PEAP proceeds to the inner authentication phase, where the client is authenticated using MS-CHAPv2 on the secure tunnel. The client starts by sending an EAP-Response/PEAP message containing its inner identity, signaling the initiation of inner authentication. In response, the PEAP server issues an EAP-Request with an MS-CHAPv2 challenge, and the client replies with an MS-CHAPv2 response. This response includes a hashed version of the client's credentials. The PEAP server then verifies the client response against its stored information, such as a hashed password or credentials located on the AAA server. If the credentials match, the client is successfully authenticated.

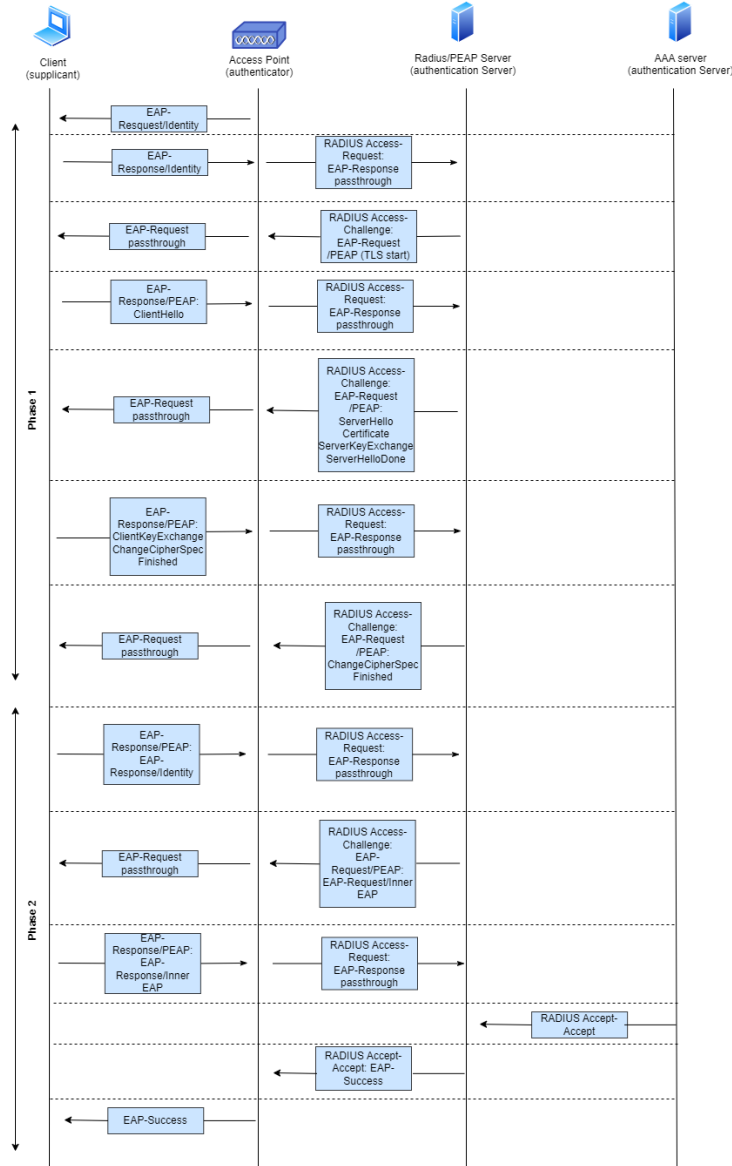


Figure 2.13: PEAP message sequences for a successful Authentication via MS-CHAPv2 based on [94]

### 2.7.4 EAP-FAST

EAP-FAST (Flexible Authentication via Secure Tunneling) [98][67] protocol is designed for secure network authentication without requiring a server-side certificate, making it faster than PKI-based methods. Instead, EAP-FAST uses Protected Access Credentials (PACs) [46] to establish a secure tunnel in Phase 1, which can then be used for authentication exchanges in Phase 2. The key components in this process are the PAC, Type-Length-Value (TLV) fields, and EAP-GTC (Generic Token Card).

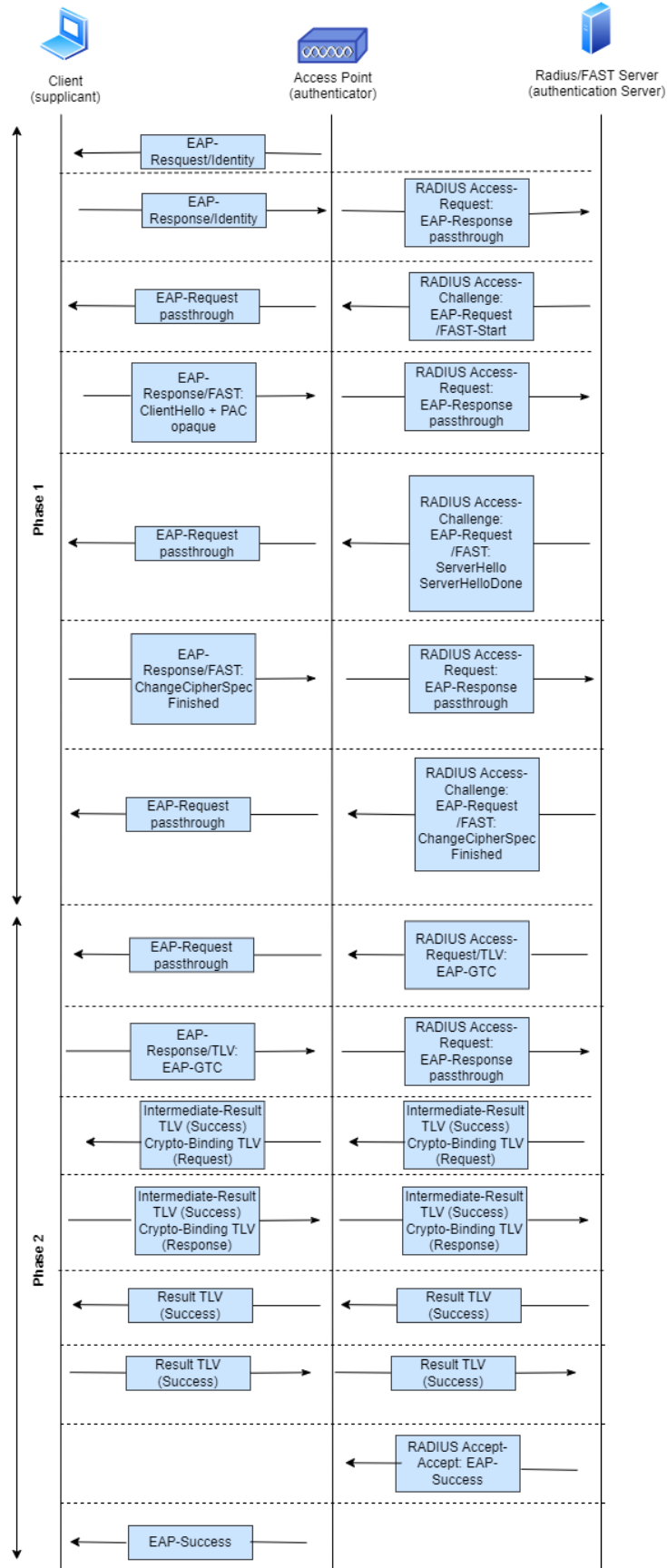


Figure 2.14: EAP-FAST message sequences for a successful Authentication based on [98]

The EAP conversation in FAST involves two phases:

- **Phase 1: Establishing a Secure Tunnel.**, the client and the server establish a secure tunnel using a PAC, which is a set of credentials shared between the client and the server. The "opaque PAC" field in the ClientHello message contains encrypted PAC data, allowing the server to verify the client's identity and establish trust without requiring certificates. This exchange ensures that the client and the server have a shared secret, enabling them to establish a secure tunnel for subsequent communications. After the PAC is exchanged, a TLS handshake occurs within the established PAC-based tunnel.
- **Phase 2: Inner authentication.** TLVs are used in Phase 2 to send different types of data in a structured way, encapsulated within the secure tunnel. For example, TLV fields can transmit attributes such as authentication tokens, cryptographic binding information, and intermediate results. Here, TLVs ensure mutual authentication and integrity by exchanging and verifying identity-related data.

EAP-GTC is a simple authentication method that is used within the secure tunnel. It allows the client to send a generic token (such as a one-time password or other credentials) to the server. This method is flexible and can accommodate different types of token-based authentication mechanism. In this example, EAP-GTC messages are passed between the client and the server to complete the authentication.

To prevent man-in-the-middle attacks, EAP-FAST uses cryptobinding TLVs, which are exchanged after the EAP-GTC phase. These TLVs verify that both the client and the server are using the same secure tunnel, binding Phase 1 (the PAC-based tunnel) and Phase 2 (authentication data) cryptographically. The successful exchange of crypto-binding TLVs confirms that both phases are linked securely.

Once the authentication and crypto-binding are complete, the server sends a result TLV (Success) to indicate successful authentication. Finally, an EAP-Success message is sent, allowing the client to access the network.

### 2.7.5 EAP-TLS

EAP-TLS [104] is similar to other EAP methods such as PEAP except that it requires authentication from both the client and the server side, and thus certificates on both sides. The process of exchanging certificates is done during the TLS handshake.

Once the initial identity exchange and the TLS handshake are done, an EAP-Success message is sent, which signals the completion of the authentication process and allows the client access to the network, as illustrated in Figure 2.15. The strength of this method is that it uses certificates both on the server and client side, making it a really secure EAP type.

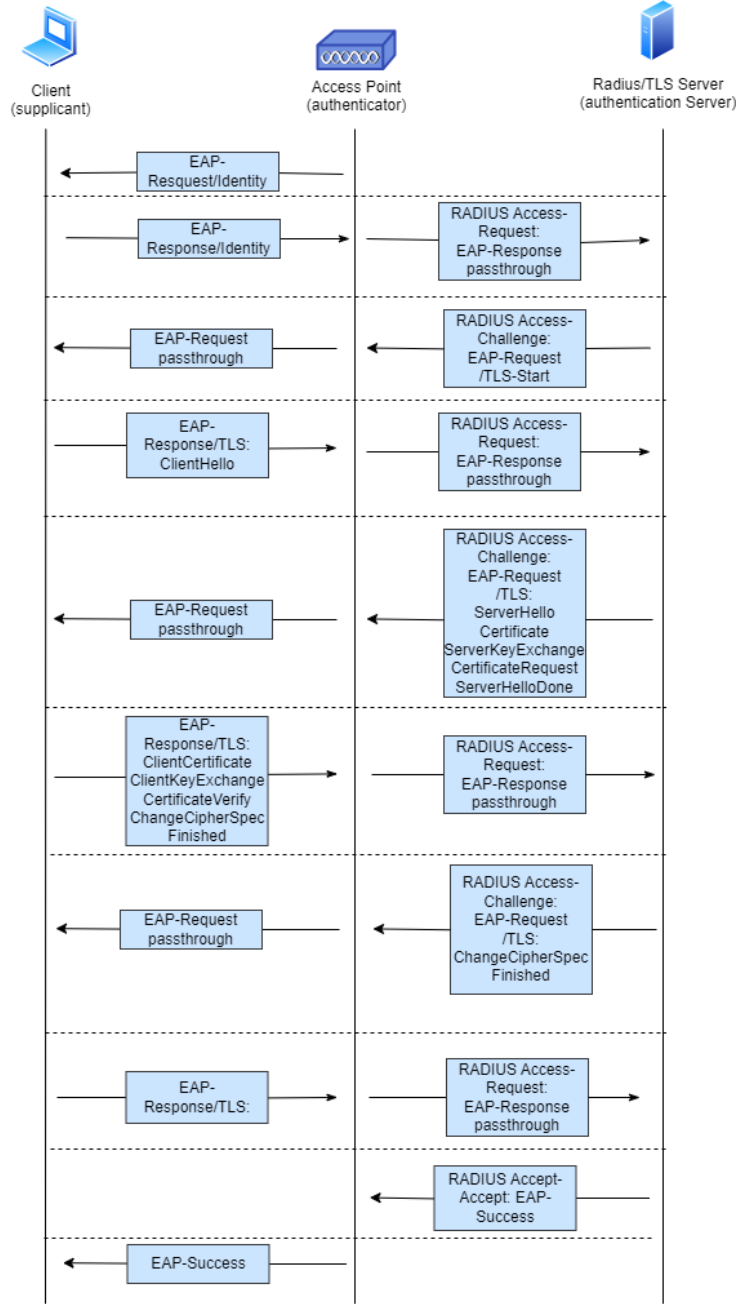


Figure 2.15: EAP-TLS message sequences for a successful Authentication based on [104]

## 2.7.6 EAP-TTLS

EAP-TTLS (Extensible Authentication Protocol-Tunneled Transport Layer Security) [45] [8] is a secure authentication method widely used in various network environments, including OpenRoaming. It is a strong method as it encapsulates a TLS session to ensure a secure and encrypted channel for the exchange of authentication information. Thus, it protects against man-in-the-middle attacks and other security threats [36]. This is particularly important in wireless environments, where data (such as password-based authentication protocols such as PAP) is transmitted over the air and can be easily intercepted.

As EAP-TTLS is often selected in the OpenRoaming context as the EAP method, it will



be covered in more depth in this section.

Figure 2.16 represents an **EAP-TTLS** message sequence for a successful authentication through tunneled **PAP**. The supplicant (client) uses **EAP-TTLS** to securely tunnel its **PAP** credentials to the authentication server via the authenticator (access point). This sequence involves the initial **EAP** identity exchange, the establishment of a **TLS** tunnel, and the transmission of credentials within the secure tunnel. **RADIUS** is the carrier protocol between the AP and the server. Both the **RADIUS** server and the **AAA** server are considered as the authentication server in this example to maintain the IEEE 802.1X three-part framework.

As soon as the client has initiated the connection, **EAP** starts with the authenticator sending an **EAP-Request/Identity** packet to the client. The authenticator forwards the **EAP-Response/Identity** to the **TTLS** server as part of a **RADIUS Access-Request** message.

**EAP-TTLS** then encapsulates the **TLS** session and the method involves 2 phases:

- **Phase 1: Establishing a Secure Tunnel (TLS Handshake).** During this phase, **TLS** is used to authenticate the **TTLS** server to the client. **EAP** packets continue to be exchanged between the client and the **TTLS** server through the authenticator (that simply passes the packets through) to complete the **TLS** handshake as presented in section 2.6.
- **Phase 2: Inner Authentication (PAP).** This phase is used to tunnel information between the client and the **TTLS** server to perform client authentication using the credentials exchanged within the secure tunnel. The authentication can itself be **EAP** or any other protocol such as **PAP**, **CHAP**, **MS-CHAP**, or **MS-CHAP-V2**. For this example, **PAP** will be used, the supplicant will thus send an **EAP-Response/TTLS** message containing its **User-Name** and **User-Password**. The authentication server validates the credentials and responds with a **RADIUS Access-Accept** message. This step ensures that the supplicant's credentials are verified against the authentication server's database.

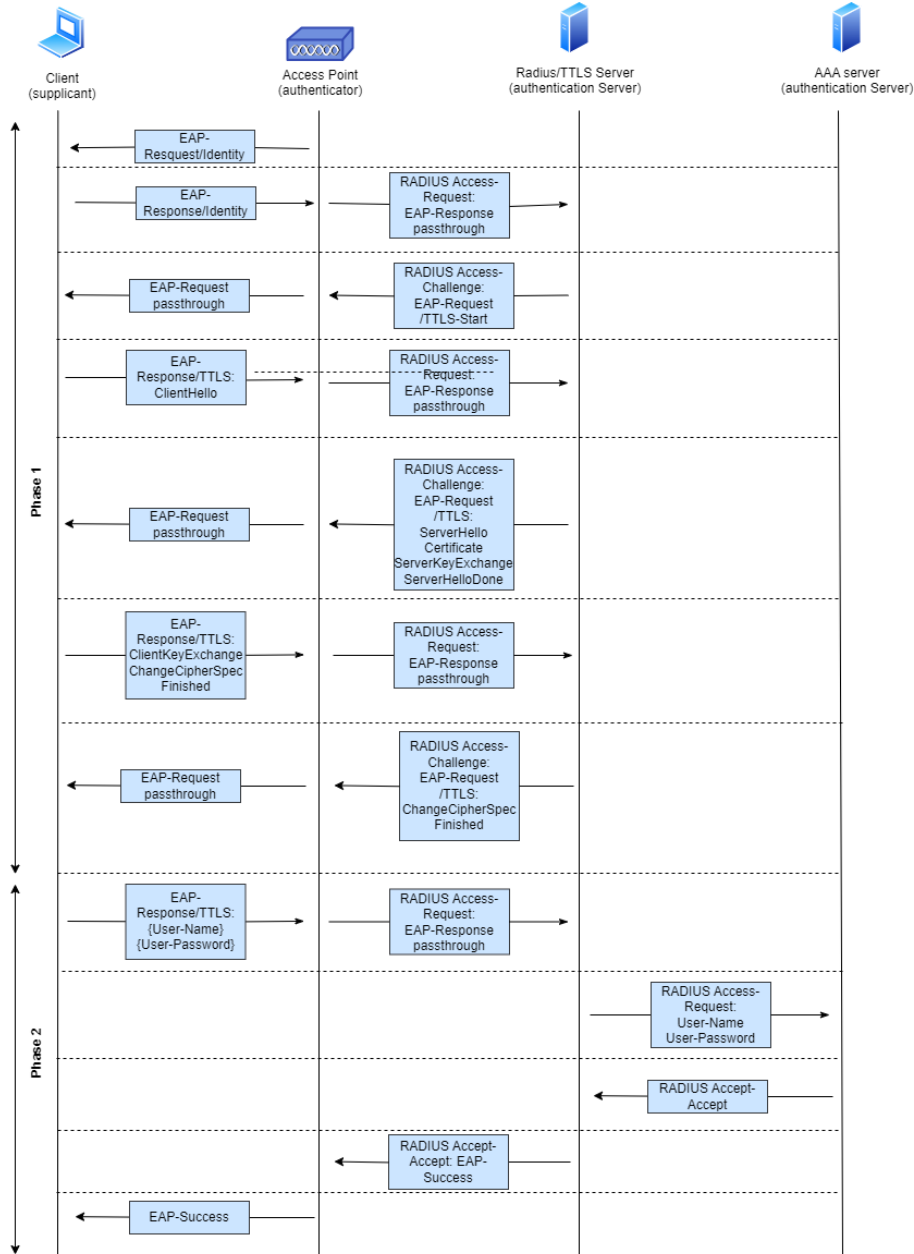


Figure 2.16: EAP-TTLS message sequences for a successful Authentication via Tunnelled PAP based on [45]

### 2.7.7 EAP-PPT

EAP-PPT (Extensible Authentication Protocol using Privacy Pass Tokens) [99] enhances user privacy and security in network authentication by preventing tracking across multiple sessions. It achieves this by using Privacy Pass tokens [92], which are unlinkable, meaning that they do not contain personal information that could identify the user or be linked to past authentication events. The Privacy Pass process works by issuing tokens (called "Privacy Pass tokens") that users can redeem on participating websites. These tokens are cryptographically signed and allow users to prove that they have passed a CAPTCHA's [128] or other challenges in the past, without revealing any personal details or requiring

the user to repeat the challenge that can compromise user anonymity.

This approach is especially interesting for scenarios where user anonymity is important, such as public Wi-Fi networks, as this will prevent service providers, identity providers and administrators to track individual's identity and personal information, or even simply connection and usage patterns.

This protocol works inside TLS tunnels and uses token challenges to authenticate peers without revealing sensitive information. As can be seen in Figure 2.17, the EAP Identity Exchange (which is optional in some cases) occurs first, where the peer sends an EAP-Response/Identity message containing only the realm portion. This allows routing to the correct authentication server without exposing any personal identifiers. The tunnel is created without the peer being authenticated initially. The server ensures that the TLS handshake does not require client certificate verification, as this is deferred in the EAP-PPT method.

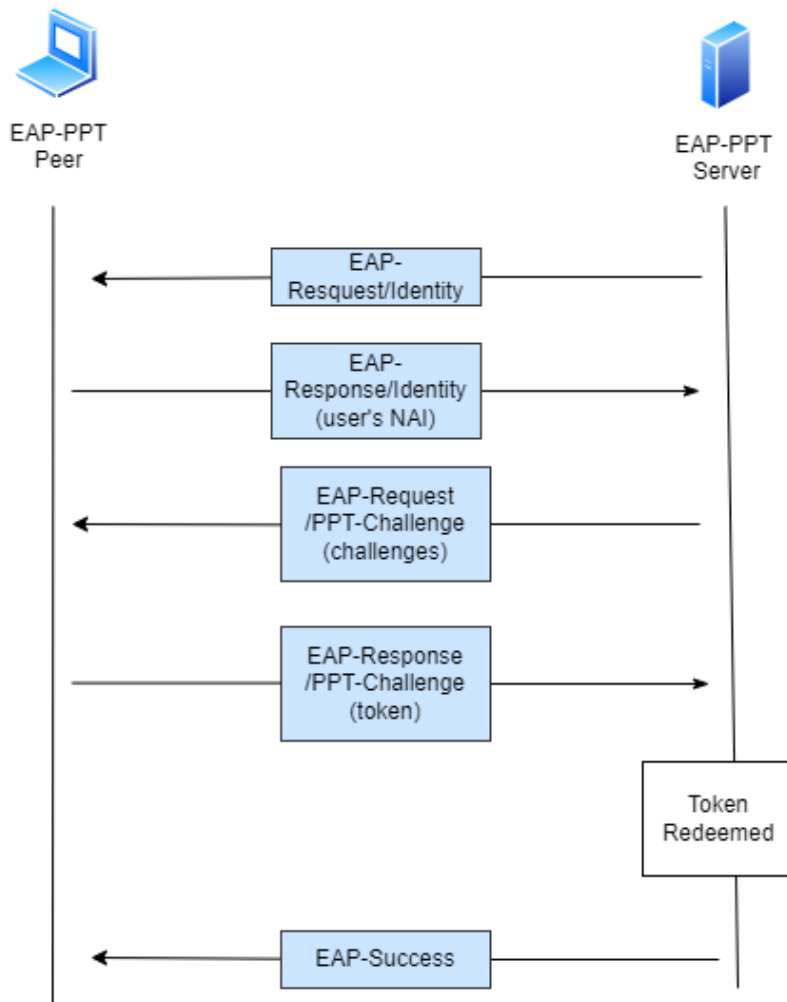


Figure 2.17: EAP-PPT message sequences for a successful Authentication taken from [99]

Once the TLS tunnel is established, the EAP server challenges the peer using an EAP-Request/PPT-Challenge message. This message contains a JSON encoded set of token challenges. These challenges require the

peer to present a valid Privacy Pass token. The peer thus sends the Privacy Pass token back to the server that can perform token verification by redeeming the provided token. If the verification is successful, the server authenticates the peer and responds with an EAP-Success message. Note that token challenges are also designed to avoid the transmission of any permanent identifiers or pseudonyms. This prevents the server from tracking the peer across different sessions.

In OpenRoaming, the realm portion of a user's outer identity helps the ANP to discover the authoritative IDP for that user. However, OpenRoaming relies on RADIUS attributes and EAP, which can lead to potential privacy concerns as user information can be shared between the ANP and IDP.

To address this issue, EAP-PPT can be used to separate the issuance of credentials from their verification, preventing the authenticator from inadvertently sharing user data. In an OpenRoaming use case, the roles of Issuer and Verifier are thus separated. The Attester/Issuer collaborates with the EAP-PPT server, which performs the token verification.

## 2.8 Wireless Broadband Alliance (WBA)

The Wireless Broadband Alliance (WBA) [135] is a global organization founded in 2003, dedicated to working with the latest Wi-Fi technologies. The organization is located in San Ramon, California, and focuses on areas such as Next Generation Wi-Fi, OpenRoaming, IoT, 5G, etc. In 2020, WBA launched the OpenRoaming federation that helps users to get a seamless and secure Wi-Fi onboarding experience.

### 2.8.1 WBA-based Public key infrastructure (PKI)

Before diving into PKI and the WBA-based PKI, it is important to introduce the asymmetric cryptography that is involved in the PKI.

#### Asymmetric cryptography

Asymmetric encryption [36], as can be seen in Figure 2.18, is used to securely send data over an insecure channel. It involves two keys: a public key and a private key. The keys are mathematically related, but it is practically impossible to derive the private key from the public key. It is thus not an issue to share the public key while keeping the other key private. One key is used for encryption and the other for decryption.

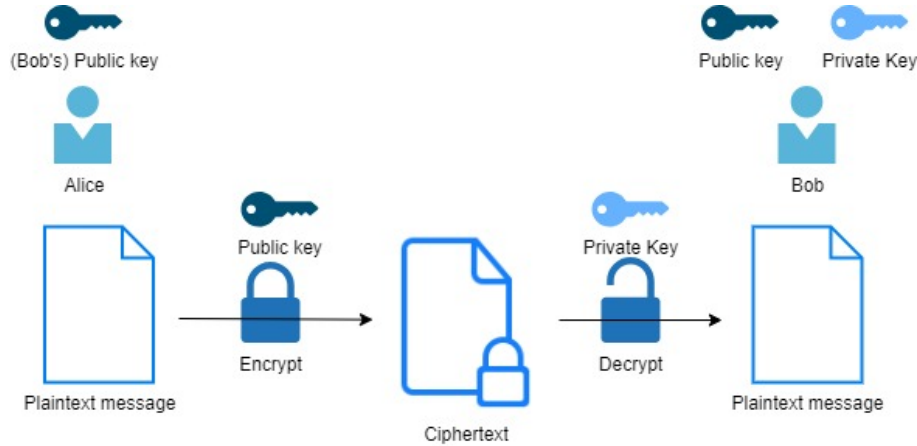


Figure 2.18: Asymmetric Encryption process based on [36]

The concept of asymmetric encryption is that Bob generates a public key and a private key using the RSA [76] algorithm. He keeps the private key secret, but shares the public key with Alice. When Alice wants to send a secure message to Bob, she uses Bob's public key to encrypt her plain text message into a cipher text. The public key can only encrypt, not decrypt. Thus, no one can read the cipher text. When Bob receives the encrypted message (cipher text), he uses his private key to decrypt it, turning the cipher text back into the original plain text. Only Bob can decrypt the message because he is the only one who has access to the private key. Thus, this process ensures secure communication between the two users.

### Public key infrastructure

To ensure that public keys are authentic (i.e. to avoid someone impersonating Bob from the previous example), a trusted third party is used for public key distribution. To do so, a public key infrastructure (PKI) model [36] is used. PKI does not distribute keys but rather certificates and the trusted third-party is called a Certification Authority (CA). Each CA holds its own certificate and the associated private key allows the CA to sign other certificates. The final form of PKI, which is hierarchical, involves a structure where there is a single root CA at the top of the hierarchy which has the authority to create and sign certificates for intermediate CAs. These intermediate CAs, in turn, can sign certificates for servers, devices, or applications. In a hierarchical PKI, the root CA's certificate is self-signed, meaning it is signed with its own private key, establishing its identity and trustworthiness. Intermediate CA's have their certificates signed by the root CA, creating a chain of trust that extends from the root CA down to the end entities. When a certificate is presented, its authenticity can be verified by tracing the chain of trust back to the root CA. This ensures that all entities within the PKI can trust each other based on this established hierarchy.

### WBA Public key infrastructure

The Wireless Broadband Alliance (WBA) Public Key Infrastructure (PKI) framework, as shown in the Figure 2.19, is a hierarchical structure designed to ensure the trust and authenticity of certificates within wireless network environments.

It operates similarly to a general PKI, with a WBA Root CA at the top of the hierarchy. openroaming.org is the root of trust and consists of a root CA, an intermediate CA and a certificate revocation service [71].

Directly below the root CA, the policy CA, which is an intermediate certificate, defines the policies related to issuing certificates within the WBA federation. There are also Intermediate Signing CA's, such as the CISCO Signing I-CA and the WBA Signing I-CA. The system also includes various agreements for the Legal Framework to ensure compliance of the ANP's and IDP's. [136]

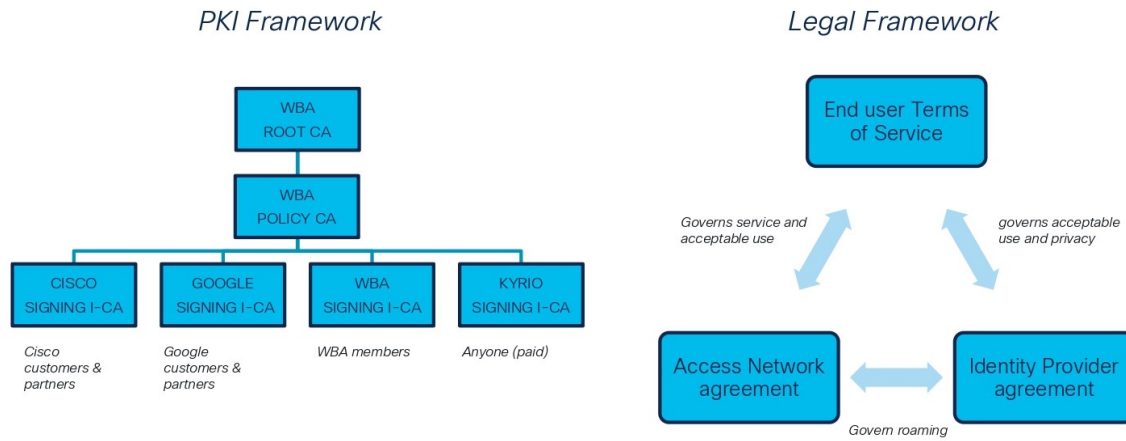


Figure 2.19: OpenRoaming federation architecture: WBA-based PKI taken from [8]

## 2.9 Passpoint

OpenRoaming leverages the Wi-Fi Alliance [125] specified Passpoint [122] [124] [58] functionality to roam. PassPoint (also known as Hotspot 2.0) allows devices to automatically connect to trusted Wi-Fi networks without the need for manual SSID selection or password entry using preconfigured PassPoint profiles. It is a secure option for roaming as it uses the 802.1X authentication standards for Wi-Fi and supports both WPA2 and WPA3 encryption. It also typically works with RADIUS servers to manage user credentials [58].

The 2 main components are the PassPoint Access Points, that can be deployed almost everywhere, and the PassPoint-enabled devices that are able to pair with the PassPoint AP's.

Passpoint-enabled AP's broadcast information about their roaming capabilities and network details (such as service providers or specific roaming partnerships) through Access Network Query Protocol (ANQP) messages, which will be discussed in the subsection 2.9.1. Devices scan for these signals, determine which networks they are authorized to join

according to the information that the AP broadcast, and connect seamlessly.

For the Passpoint-enabled devices to "know" which networks they are authorized to connect to, Passpoint uses preconfigured profiles that are installed on the devices. These profiles contain the necessary network and EAP credential information required to authenticate the user seamlessly.

During the user's authentication, as sensitive information is going to be exchanged, the connection is protected by utilizing the IEEE 802.1X standard along with WPA2 or WPA3 encryption protocols. The user's credentials are managed by a RADIUS/RADSEC server which supports multiple credential types and is secure.

Passpoints are thus a core component of the **OpenRoaming** infrastructure as they enable seamless roaming by allowing users to move between different networks and access points without disconnection or re-authentication.

### 2.9.1 Access Network Query Protocol (ANQP)

The Access Network Query Protocol (ANQP) [136] [112] is a protocol used by wireless devices in wireless networks to get information about available Wi-Fi networks before actually connecting to them. ANQP uses active scanning [70] as the client broadcasts a Probe Request frame and waits for the Probe Responses from the nearby AP's. The client can then get access to information about the AP's and their capabilities. It can include information about capabilities, authentication supported, Roaming Consortium, Hotspot 2.0 information, etc.

In the context of **OpenRoaming**, this protocol is particularly relevant as it provides information on Hotspot 2.0 and Roaming Consortium, which enables users to establish a connection automatically.

### 2.9.2 Roaming Consortium Organization Identifier (RCOI)

An ANP need to be able to set up a policy in order to accept or reject users to its network. To do so, Closed Access Group (CAG) based policies are used. Users who are part of a CAG are allowed to access one or more **OpenRoaming** access networks, depending on the CAG policy and what it allows. Roaming Consortium Organization Identifier (RCOI) [136] [113] is used to encode these CAG's.

RCOI is a 24 or 36-bit identifier used within the **OpenRoaming** Passpoint profile. It manages the identity providers that are supported and allowed by the network.

There are 2 parts in the RCOI, as shown on Figure 2.20:

- **The base RCOI:** the first 24 bits. There are 2 possible base RCOI's:
  1. OpenRoaming-Settled: BA-A2-D0-xx-x which indicates that the users need to pay the ANP to use their **OpenRoaming** services.
  2. OpenRoaming-Settlement-Free: 5A-03-BA-xx-x which indicates that the ANP is free of use.

- **The CAG extension:** the last 12 bits, used to implement the Closed Access Group Policies (CAG) and defined by the WBA. There are 5 fields, as can be seen on Figure 2.20:
1. Level of Assurance Policies (LoA) [85]: It defines the degree of confidence during authentication and security levels for identity proofing, with values for “Baseline” (LoA 2) and “Enhanced” (LoA 3).
  2. Quality of Service Policies (QoS) [131]: It sets service quality levels (Bronze and Silver) to ensure consistent network performance. QoS provides guarantees on availability, basic bandwidth and other performances of a service such as packet loss, latency, etc.
  3. Privacy Policies (PID): It ensures the identities of the users remain anonymous through EAP identifiers that mask user information. The PID field enables tracking if users consent to share a unique identifier, allowing an ANP to analyze usage patterns and other information. It thus states if a user will be anonymous or have personal identity.
  4. ID-Type Policies (ID-Type): It specifies sectors (e.g., any, government, education, retail) using an ID-Type field, allowing the ANP’s to serve users based on sector-specific requirements.
  5. Reserved: The last 4 bits are set to 0.

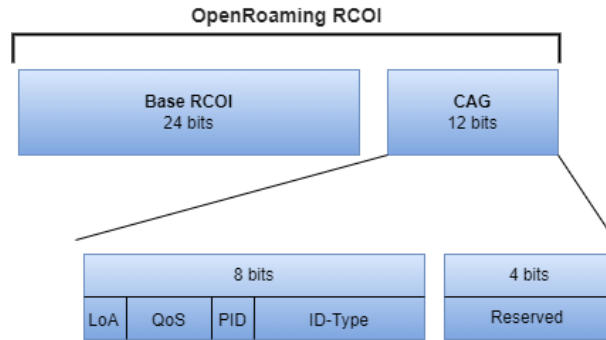


Figure 2.20: OpenRoaming RCOI format based on [136] [113]

The RCOI’s are included in the OpenRoaming Passpoint profiles on the device and broadcasted via the beacons. The beacon can only carry 3 RCOI’s. If there is more than 3, the ANQP protocol is used to advertise larger lists of RCOI’s. In the case where an RCOI matches a configured profile in the device, the device will then start the authentication process. A matching RCOI is thus necessary to access a network, as depicted in Figure 2.21.



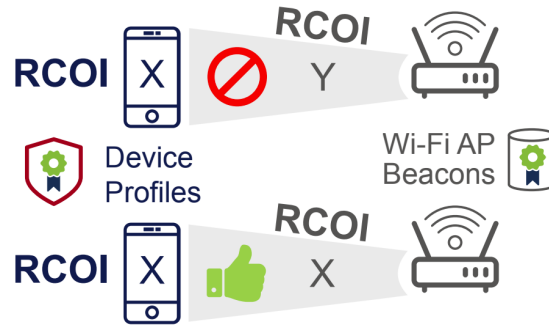


Figure 2.21: IDP profile and ANP matching scenarios taken from [113]

Regarding the link with ANP and IDP, the ANP will thus need to configure the RCOI in its Wi-Fi equipment while the IDP will need to include the RCOI in the Passpoint profile of its user devices. The RCOI in the profile and the one in the Wi-Fi equipment need to match.

The ANP will state in the RCOI what it requires as PID and LoA and what it provides as QoS. The IDP will state what it provides as PID and LoA and what it requires as QoS. It means that if the RCOI matches, all requirements from both parts are met.

In the case where an ANP does not want to authorize all users from a CAG encoded using RCOI, it can use a list of authorized NAI realm instead.

# Chapter 3

## Theoretical Background: e-ID

This chapter describes **e-ID** and related technologies in more details to provide a foundation for the subsequent phases of this project.

### 3.1 Electronic identification (e-ID)

**e-ID** [42] [41] is an authentication method. It authenticates users using an electronic proof of identity via the chip on the Belgian electronic identity card.

The e-ID chip typically contains encrypted personal information, such as name, birth date, and sometimes biometric data, which are securely stored on a chip. Digital certificates can also be installed by the Belgian Government on the electronic chip if the card is **e-ID-capable**. These digital certificates are used to authenticate the card holder digitally. A PKI is commonly used in e-ID systems, where digital certificates help authenticate the user and protect communication between the user and service providers [38] [14].

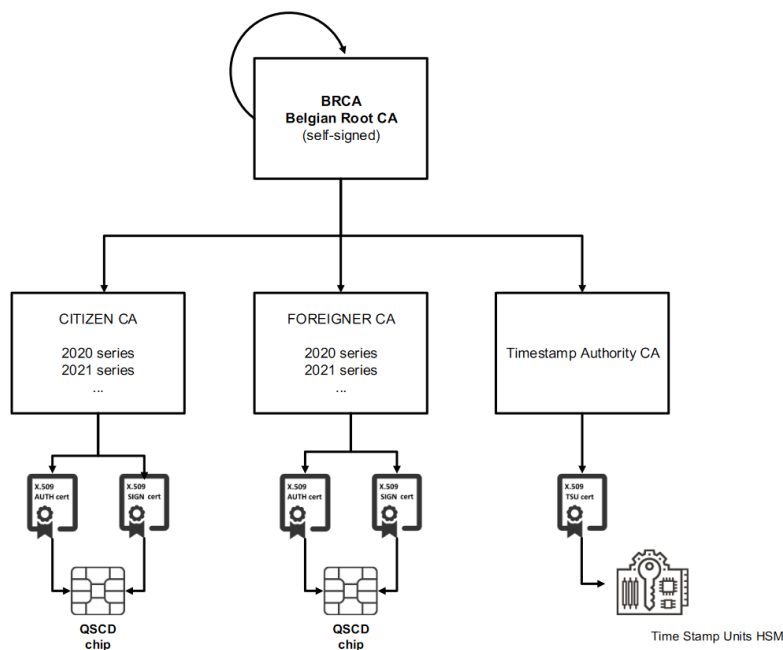


Figure 3.1: e-ID-based PKI taken from [38]

The Figure 3.1 illustrates the Public Key Infrastructure (PKI) hierarchy used for Belgian e-ID cards. The Belgian Root CA (BRCA) is the top-level CA managed by the Belgian government. It issues certificates to intermediate CAs. There are 3 intermediate CAs that the BRCA directly certifies. The Citizen CA and Foreigner CA issue certificates for Belgian citizens and foreigners. The timestamp Authority CA issue timestamp certificates, which are used to provide a secure timestamp for digital signatures. This ensures that the exact time and date of a signature or transaction can be verified. Each intermediate CA issues certificates to end-users or devices that are, in this case, embedded in chips or specific hardware. Citizen CA and Foreigner CA issue two main types of certificates: the X.509 AUTH cert, that is used for authentication purposes, allowing a cardholder to verify its identity, and the X.509 SIGN cert, used for digital signing. These certificates are embedded in Qualified Signature Creation Device (QSCD) [39] chips. The QSCD is a secure environment where private keys are stored, ensuring that the digital signature is created under strict security controls.

The CA's issue personalized certificates for individual electronic chip cards. These certificates leverage X.509, which is a standard that defines the format of the public-key certificates [43]. X.509 certificates bind a public key with information about the entity (such as name, organization, location, signature algorithm, etc.) and is signed by a trusted CA, making it verifiable by anyone who trusts that CA.

### 3.1.1 Multi-factor Authentication

To improve its security, the e-ID system implements a strict authentication protocol. For example, PIN-based access combined with PKI-based digital certificates ensures only authorized users can access the e-ID.

This process is called a Multi-factor Authentication (MFA) [74]. MFA is a security process that strengthens user verification by requiring multiple separate categories of identification to access a system or application. These categories are:

- **Something You Know:** This category includes credentials such as passwords, PINs, or answers to security questions.
- **Something You Have:** This might be a mobile device, a security token, or a smart card.
- **Something You Are:** This can be biometrics, such as fingerprints, facial recognition, or voice verification.

Usually, a user is prompted for 2 of these 3 categories to prove its identity. This is called 2-factor authentication (2FA). e-ID can work using "Something You Know" and "Something You Have". Indeed, many e-IDs are smart cards that need to be inserted into a reader. The user then enters a PIN to verify access, which acts as the second factor.

The e-ID can also be used to bootstrap 2-factor authentication using applications such as myGov [109] or Itsme [11].

## 3.2 OAuth 2.0 protocol and OpenID Connect (OIDC)

The e-ID identity management solution supports OpenID Connect (OIDC) [90]. OIDC is an identity layer built on top of OAuth 2.0 [57] [89], designed for user authentication. Let us dive into these protocols [26].

### 3.2.1 OAuth 2.0 protocol

Before OAuth 2.0, if an application wanted to access another service on behalf of a user, it would often ask for the user's password to that service. This means that the user was giving the third-party applications access to everything, regardless of what specific permissions the third-party application actually needed. This also means that if one application is compromised, all accounts where that password was used could be at risk.

OAuth 2.0, which stands for Open Authorization is a protocol designed to provide delegated authority that enables third-party applications to obtain limited access to their data without sharing passwords. Instead, it uses access tokens, which are generated by an authorization server and grant specific permissions to the third-party app to retrieve specific user data or get access to remote API's.

#### OAuth 2.0 authorization flow

The Figure 3.2 illustrates the OAuth 2.0 authorization flow and the interactions between the four primary roles in the protocol.

The four roles are:

1. **The user (Resource Owner):** This is the person who owns the data or resource that the third-party application (client) wants to access. The user decides whether to grant the client permission to access their data.
2. **The client (Third-Party Application):** The client is the application that wants to access the user's data on another server.
3. **The authorization server:** This server is responsible for authenticating the user and determining whether the user grants permission to the client or not. It issues an access token if the user authorizes the request.
4. **The resource server:** The resource server hosts the protected resources (e.g., user's data) that the client wants to access. The authorization server and the resource server can be a unique server which manages the 2 roles.

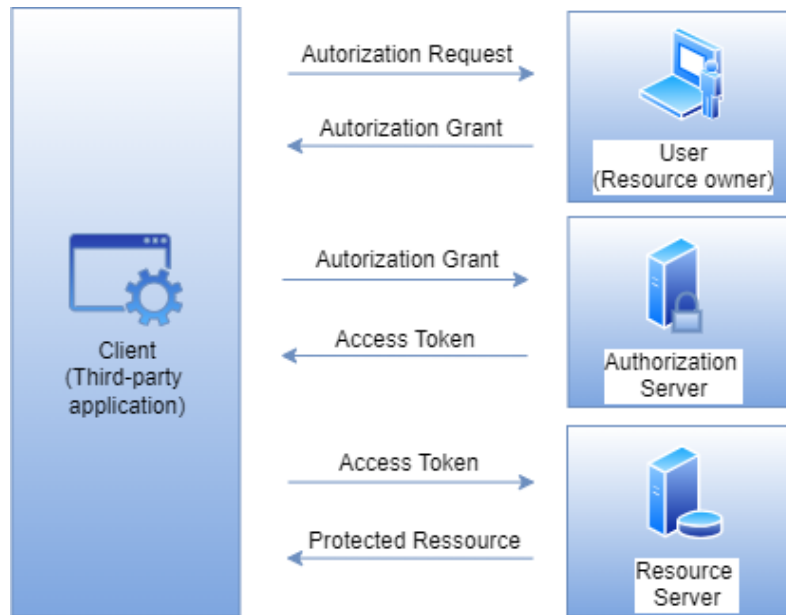


Figure 3.2: OAuth 2.0 flow and the interaction between the four roles based on [57]

The client begins the flow with the user being asked to log in and grant the client permission to access their resources. Note that the request can be made directly to the resource owner or via the authorization server as an intermediary. This request includes information such as the client ID (to uniquely identify the client), requested scopes (permissions), and a redirect URI of the client that indicates where to send the authorization grant. An authorization grant is a kind of credential that represents the user's authorization.

The client then sends the authorization grant to the authorization server's token endpoint to request an access token. This step includes the client's credentials (client ID and secret) and the authorization grant, ensuring that only authorized clients can exchange the grant for a token. The authorization server validates the authorization grant and, if successful, issues an access token to the client. The access token represents the client's authorized access to the user's resources and typically expires after some time.

When the current access token expires, refresh tokens are used to obtain new access tokens. Refresh tokens are issued to the client by the authorization server, usually at the same time as the access tokens. The way in which access and refresh tokens are used is shown in Figure 3.3. The client exchanges the authorization grant with the authorization server to obtain an access token along with a refresh token. The access token allows the client to access the user's protected resources on the resource server while the refresh token allows the client to obtain a new access token after the original one expires, without requiring the user to authenticate again. If the access token is valid, the resource server grants the client access to the protected resource and returns the requested data. Access tokens expire after some time, so when the client attempts to use an expired or invalid access token to access the resource server, the resource server responds with an error, indicating that the token is invalid. Instead of asking the user to re-authenticate and start again the whole OAuth 2.0 flow, the client uses the refresh token to request a new access

token from the authorization server.

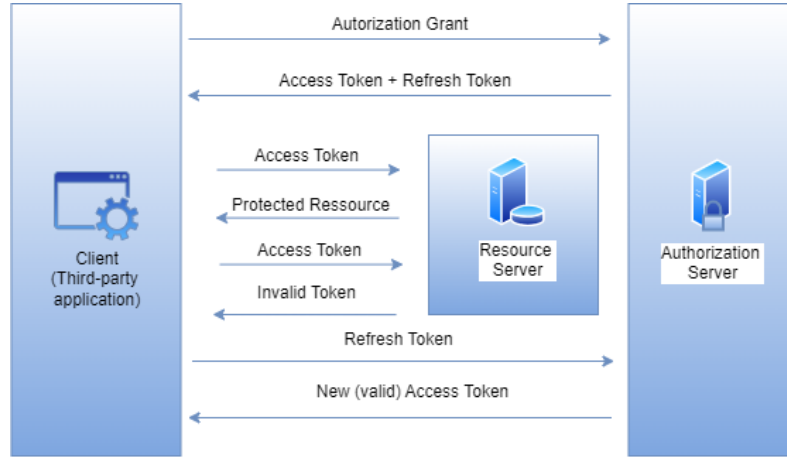


Figure 3.3: Tokens usage based on [57]

Finally, the resource server validates the access token and provides the requested data to the client if the token is valid and authorized for the requested scope.

### Client Registration <sup>1</sup>

For the OAuth 2.0 authorization flow to work, the client needs to register with the authorization server. During registration, the client provides details such as its name, type, and redirect URI (the URL to which the authorization server will send the user after authorization). After registration, the authorization server issues a unique client identifier to the client. This client ID is public and used to identify the client during the authorization process. In addition to the client ID, the authorization server may issue a client secret (a password-like credential) to the client. The client secret is used to authenticate the client when it requests an access token.

### OAuth 2.0 Endpoints

The Authorization Code Flow involves three primary protocol endpoints that enable communication between the client and the authorization server. These endpoints are typically exposed via HTTP/1.1 [88]. Note that OAuth 2.0 typically requires HTTPS for all requests to ensure that the client ID, client secret, and tokens are transmitted securely. It thus uses REST calls for the requests. The 3 endpoints are:

- **The authorization/authentication endpoint:** This is where the client directs the user to initiate the authorization process. This endpoint enables the user to grant permission to the client to access their resources. The authorization server

<sup>1</sup>This is the step that made the integration with e-ID impossible to demonstrate in this project, because BOSA was responsible for providing this client id and secret but I didn't receive them in time. On the other hand, Google allows you to register yourself, and this allowed the project to continue without any major changes.

authenticates the user, obtains consent, and then redirects the user back to the client with an authorization code.

- **The token endpoint:** This is where the client exchanges the authorization code for an access token and possibly a refresh token. The client must authenticate itself (typically using its client ID and client secret) when making this request.
- **The redirect URI:** This is not a dedicated endpoint but an endpoint within the client application that receives the authorization code. The client registers this URI with the authorization server, so the server knows where to send the user back after authorization.

### 3.2.2 OpenID Connect (OIDC)

While OAuth 2.0 was primarily used for authorization (granting access to resources without sharing passwords), it does not provide a standardized way to verify the user's identity and to define the scopes. To address the limitation of OAuth 2.0 for authentication, OpenID Connect (OIDC) [90] was developed to standardize the user's authentication.

OIDC [90] is a standard built on top of OAuth 2.0. In a typical OAuth 2.0 flow, the authorization server issues an access token and possibly a refresh token to the client. However, these tokens only allow the client to access resources on behalf of the user and they do not contain information about the user's identity. OIDC introduces an Identity (ID) token to address this. The ID token contains claims (which are key-value pairs containing information) about the authenticated user, such as their unique identifier, name, and email address and allows the client to verify the user's identity. The token format required for the ID token is a JSON Web Token (JWT) [64]. In OAuth 2.0, multiple token formats are accepted, including JWT.

By adding this OIDC standard, when a client thus asks the authorization server for tokens, it will get an access token, a refresh token and an ID token that uniquely identifies the user.

OIDC adds a new endpoint to the standard OAuth 2.0 endpoints, the UserInfo Endpoint, that returns claims about the authenticated user.

Another limitation that OIDC addresses is the fact that OAuth 2.0 does not specify which scopes to use for authentication, leaving it up to clients to define custom scopes. OIDC addresses this issue by introducing standardized scopes, such as openid, profile, and email. The openid scope is required in any OIDC authentication request and indicates that the client wants to use OpenID Connect for user authentication.

By adding an authentication layer, OIDC is considered as an Identity Provider (IDP), allowing clients to verify a user's identity in a standardized way.

### 3.2.3 Tokens format

OpenID Connect uses the JSON Web Token (JWT) [64] [7]. JWT is a standard used to exchange claims between two side, such as a client and a server. Claims [28] are pieces of

information asserted about a subject, typically a user, and are represented as key-value pairs within the token. In JWT, an example of claim about a user's name would be:

*"name" : "John Doe"*

The structure of a JWT consists of three parts, separated by dots, as can be seen in Figure 3.4

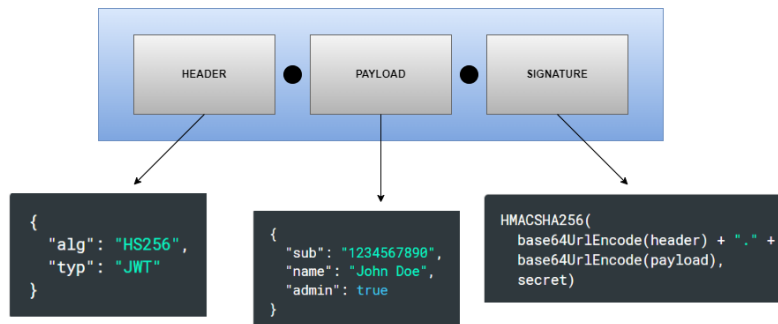


Figure 3.4: structure of a JWT based on [7]

The header typically consists of two fields, the type of the token (JWT) and the signing algorithm used (e.g., HS256 for HMAC SHA-256 and RS256 for RSA SHA-256).

The payload, which is Base64Url [66] encoded, contains the claims. There are three types of claims:

- Registered Claims: Predefined claims that provide metadata, such as iss (issuer), sub (subject) and exp (expiration time).
- Public Claims: Custom claims such as name or email.
- Private Claims: Claims agreed upon by the two parties but not standardized.

The signature is created by taking the encoded header and encoded payload, joining them with a dot, and signing them with a secret key using the algorithm specified in the header.

With OpenID Connect, when a user logs in, the server generates a JWT, signs it, and returns it to the client. The client can then include it in the authorization header of future requests using the Bearer schema. This header is part of the HTTP standard for sending authentication information and has the following format [65]:

```

GET /resource HTTP/1.1
Host: server.example.com
Authorization: Bearer <token>

```



# Chapter 4

## Solution investigation

### 4.1 Context for this project

In this project, the goal is to use **e-ID**<sup>1</sup> as an IDP. To implement that, several components need to be considered. Figure 4.1 illustrates these components and each of them will be described in more detail in the following sections.

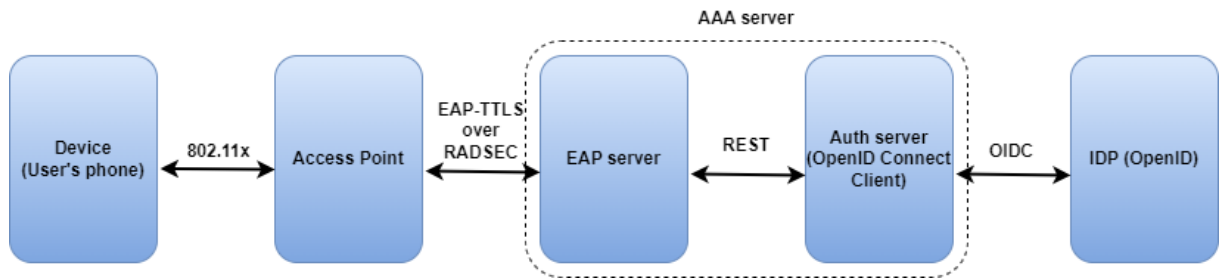


Figure 4.1: OpenRoaming: Components of the project

### 4.2 Device (User's phone)

On the user's phone, a client application will be developed and installed. This application should allow the user to do two actions:

- The user should be able to authenticate with the IDP when opening the application through an authentication screen. If successful, the user will receive tokens from the **e-ID** IDP that will be used for the **OpenRoaming** profile. This part is thus similar to applications that require national authentication such as MyGov.be [109], ItsMe [11], MyPension [84], etc. The application developed for this project will thus be a mock-up of these applications with a feature that allows to provision users with **e-ID** tokens when they authenticate.

---

<sup>1</sup>Or Firebase with Google credentials, as the process is exactly the same. I mention this because the integration with **e-ID** was not possible as BOSA was responsible for registering me as one of their clients but it was not done in time.

- When authenticated, it should allow the device to seamlessly connect to the Wi-Fi via an 802.1x connection to the access point. This is done by installing a profile on the device. These pre-configured Passpoint profiles allow the devices to automatically connect to a trusted Wi-Fi without the need to select the right SSID and manually enter credentials.

### 4.2.1 Programming language selection

To choose a programming language for this application, some aspects of the application needed to be taken into account.

First, a cross-platform language, that allows the same application to run on both iOS and Android, was not an option. Indeed, the application will need to interact with device-specific aspects such as the device parameters and Wi-Fi settings. Thus, a cross-platform language such as Flutter might complicate access to these functionalities.

Since I have a Windows laptop, focusing on Android development is practical, as developing and deploying iOS applications with Swift generally requires a Mac from Apple. Note that developing with Swift for iOS on a Windows computer is possible, but it is generally more complicated because Xcode, the official Apple IDE for Swift development, is only available on macOS.

The final choice moves towards Android development. Kotlin is now the main language recommended and supported for developing Android applications [53], so this is the language chosen for this project. To develop this application, an object-oriented approach will be used.

### 4.2.2 Passpoint profile

This is the template [6] [35] for Android devices using username/password as credentials, which is the method used for EAP-TTLS with PAP:

```

1 <MgmtTree xmlns="syncml:dmddf1.2">
2   <VerDTD>1.2</VerDTD>
3   <Node>
4     <NodeName>PerProviderSubscription</NodeName>
5     <RTProperties>
6       <Type>
7         <DDFName>urn:wfa:mo:hotspot2dot0 -
          perprovidersubscription:1.0</DDFName>
8       </Type>
9     </RTProperties>
10    <Node>
11      <NodeName>i001</NodeName>
12      <Node>
13        <NodeName>HomeSP</NodeName>

```

```

14         <Node>
15             <NodeName>FriendlyName</NodeName>
16             <Value>${friendlyName}</Value>
17         </Node>
18         <Node>
19             <NodeName>FQDN</NodeName>
20             <Value>${fqdn}</Value>
21         </Node>
22         <Node>
23             <NodeName>RoamingConsortiumOI</NodeName>
24             <Value>004096</Value>
25         </Node>
26     </Node>
27     <Node>
28         <NodeName>Credential</NodeName>
29         <Node>
30             <NodeName>Realm</NodeName>
31             <Value>${realm}</Value>
32         </Node>
33         <Node>
34             <NodeName>UsernamePassword</NodeName>
35             <Node>
36                 <NodeName>Username</NodeName>
37                 <Value>${username}</Value>
38             </Node>
39             <Node>
40                 <NodeName>Password</NodeName>
41                 <Value>${password}</Value>
42             </Node>
43             <Node>
44                 <NodeName>EAPMethod</NodeName>
45                 <Node>
46                     <NodeName>EAPType</NodeName>
47                     <Value>${eapType}</Value>
48                 </Node>
49                 <Node>
50                     <NodeName>InnerMethod</NodeName>
51                     <Value>${innerMethod}</Value>
52                 </Node>
53             </Node>
54         </Node>
55     </Node>
56 </Node>
57 </Node>

```

58 `</MgmtTree>`

Listing 4.1: Profile with a username/password credential (EAP-TTLS) taken from [6] [35]

This profile will be downloaded from a server (the auth server) and will allow a device to seamlessly connect to the Wi-Fi using this profile. The profile will include hard-coded values, such as the FQDN of the server (*marie.tiedie.io*), the realm (*test-beid.openroaming.net*), the EAP type and the inner method according to the EAP type. The RCOI is defined with the default Cisco value 004096 to accept users from any IDP. It will also contain user-specific values such as the username and the password, which is base64-encoded [6].

## 4.3 Access Point

Modern access points support advanced features such as load balancing, seamless roaming, and better management. They are essential for Wi-Fi configuration, which is where Meraki access points come in, adding cloud management and additional capabilities for organizations.

### 4.3.1 Meraki

Cisco Meraki <sup>2</sup> [21] is a cloud-managed network solutions provider that provides Wi-Fi 6/6E hardware.

The Meraki access points are wireless devices that enable seamless Wi-Fi. These access points are cloud-managed and can be easily deployed and managed through the Meraki dashboard [20].

The Meraki Dashboard is a platform that allows one to manage an entire Meraki network from the interface shown in Figure 4.4. Especially, Figure 4.2 shows the homepage of the dashboard. It summarizes the available access points, the usage and the clients along with data about them. Figure 4.3 shows an overview of the configured SSIDs and their characteristics. The Wi-Fi settings of the computer that is near the access point display the 3 SSIDs configured: Marie network - wireless Wi-Fi, Mamaes and Test1. More advanced settings are available, such as a configuration page to set up access control for a SSID. This will be discussed in chapter 5.

The Meraki access point is a Wi-Fi 6/6E hardware. Wi-Fi 6 and Wi-Fi 6E standards are the latest generations of Wi-Fi technology, based on the IEEE 802.11ax standard. It improves the previous Wi-Fi generations by providing higher throughput, lower latency, and more efficient traffic management.

<sup>2</sup>Meaning of the greek word: Meraki • *Μεράκι* [may - rah - kee] - (adj.) when you do something with soul, creativity or love; putting a piece of yourself into what you do. [75]

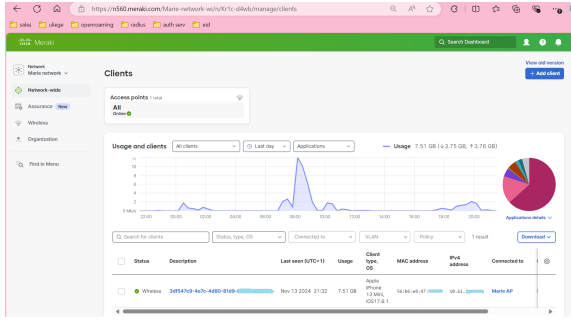


Figure 4.2: Home page: summary of the clients

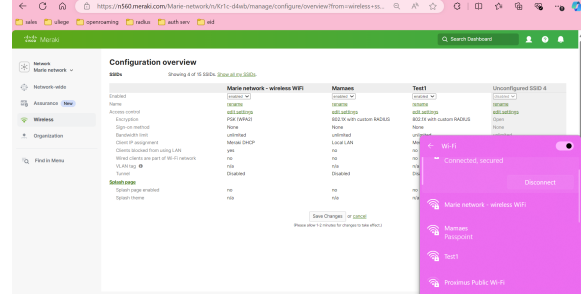


Figure 4.3: SSID overview with their configuration and computer Wi-Fi settings displaying them

Figure 4.4: Meraki dashboard overview [20]

### 4.3.2 OpenRoaming considerations: Cisco Spaces

Once Meraki access points are set up to manage Wi-Fi networks, Cisco Spaces [15] can be introduced to enable **OpenRoaming**. Cisco Spaces provides the tools needed to provide automatic, secure connections to Wi-Fi networks without requiring users to log in every time. Working together with Meraki, it helps create a seamless connection where users can easily move between trusted networks. To set up **OpenRoaming** through Cisco Spaces, a space account is thus needed and the cloud-based (Cisco Meraki) network is supported. The wireless network then needs to be added to the Cisco Spaces account. This will be explained in chapter 5.

## 4.4 RADIUS Server

The RADIUS server must be set up to combine EAP with RADIUS within the AAA server. Its purpose is to ensure that only authorized users can access the network by approving or declining the requests coming from the AP.

As the server needs to be public to receive requests when users want to get access to a network, a public server will be used in this project with the name *marie.tiedie.io* and the IP address 185.48.12.253.

### 4.4.1 Technologies used

Creating a RADIUS server from scratch is quite challenging. An open-source option is thus a good solution. FreeRADIUS [118] is a popular choice for an EAP/RADIUS server and is used in well-known applications such as eduroam [3]. FreeRADIUS is open-source and supports multiple EAP types, including EAP-TLS, EAP-TTLS, PEAP, and others. It also supports secure communication protocols such as RADSEC (RADIUS over TLS) and robust certificate management for EAP-TLS and EAP-TTLS. Moreover, FreeRADIUS allows us to add custom modules or plugins to extend its functionalities. For example, a REST module and a EAP module are available and will be used for this project. Indeed, EAP is used to

communicate with the access point while a RESTful interface will be used to make requests to the OpenID Connect Auth server.

#### 4.4.2 EAP method selection

##### EAP method

To choose the right EAP method [100], Table 4.1 compare some of the most common EAP types. The table analyzes factors such as security, implementation complexity, compatibility with OpenRoaming requirements, and support for the authentication method. The goal is to find an EAP type that is secure and compatible with both FreeRADIUS and the e-ID credentials distribution.

	LEAP	PEAP	EAP-FAST	EAP-TLS	EAP-TTLS	EAP-PPT
Acronym	Lightweight EAP	Protected EAP	Flexible Authentication via Secure Tunneling	EAP Transport Layer Security	EAP Tunneled Transport Layer Security	EAP Privacy Pass Token
Developed By	Cisco	Microsoft and Cisco	Cisco	IETF	Funk	Cisco
Security	Poor	Moderate	High (requires PAC files)	Very High (client and server certificates)	High (server certificate with optional client certificates)	Very High
Authentication Method	Username/Password	Username/Password in a TLS tunnel	PAC file with username/password	Certificate-based (both client and server)	Username/Password in a TLS tunnel	Privacy Pass Token in a TLS tunnel
Encryption	WEP (weak)	TLS	Tunnel established through TLS	TLS	TLS	TLS
Protected Tunnel	No	Yes	Yes	Yes	Yes	Yes
Vulnerability to Attacks	High	Moderate to High, depends on server certificate security	Vulnerable if PACs are compromised	Low, very secure if certificates are managed properly	Low to Moderate, secure if implemented correctly	Low, very secure
Certificate Requirement	No	Server certificate	Server certificate (for PAC provisioning)	Client and server certificates	Server certificate (optional client certificate)	Server certificate and NO client certificate (anonymous client)
Ease of Deployment	Simple	Moderate	Moderate	Complex	Moderate	Complex
Use Cases	non-critical applications	Enterprise Wi-Fi	Environments needing good security without certificates	High-security environments needing strong mutual authentication	Enterprise Wi-Fi with both flexibility and strong security	Access to restricted services requiring anonymous client authorization
Fast Reconnect	No	Yes	Yes	Yes	Yes	Yes
Scalability	Moderate	High	High	High	High	High

Table 4.1: EAP types comparison

FreeRADIUS only supports LEAP, PEAP, EAP-FAST, EAP-TLS and EAP-TTLS as the EAP type [114]. However, although EAP-FAST is supported by FreeRADIUS, it has several set-up limitations since FreeRADIUS does not provide tools and features for automated

PAC management. As a result, EAP-PPT and EAP-FAST were not selected, as they are either unsupported or require huge modifications to integrate with FreeRADIUS.

Unsurprisingly, FreeRADIUS does not recommend using LEAP in new deployments as it has serious security issues, including weak encryption mechanisms and susceptibility to offline password attacks.

PEAP, which is more secure than LEAP due to its encapsulation within a TLS tunnel, is still not secure enough for an e-ID integration. Indeed, it is less robust than other protocols such as EAP-TLS or EAP-TTLS. Thus, it is not a good choice for this project as security is a major concern.

Finally, EAP-TTLS was chosen over EAP-TLS mainly because OpenRoaming requires the use of credentials (in this project, the credentials are the tokens) that are generated by the IDP after a successful authentication. EAP-TLS, while highly secure due to its use of mutual certificate authentication, only works with certificates for client authentication. This makes integration difficult with e-ID as e-ID credentials are tokens that are specific to each user.

In contrast, EAP-TTLS allows the use of username/password credentials within an encrypted tunnel, making it a great choice for the OpenRoaming framework, as the IDP issues tokens when a user has successfully authenticated.

EAP-TTLS is thus the final type chosen for this project. [137]

### Inner EAP method

PAP was selected as the inner authentication method for EAP-TTLS because it works well with token credentials and is usually the inner method selected when EAP-TTLS is the EAP type. PAP transmits a username/password pair and in this project, the username will be the ID token and the password will be the access token.

Since a secure TLS tunnel is established before the exchange of credentials, security is already ensured and thus PAP is enough to transmit them. It is also possible to avoid transmitting the actual plaintext credentials by hashing them. The backend server will then compare an hashed version of the access token to see if the user is authorized. [93]

## 4.5 Auth server

### 4.5.1 Programming language selection

As Kotlin is used for the mobile application on the device, it can also be used to code this server to remain consistent throughout the project in terms of the programming language used.

On top of that, Kotlin supports frameworks like Ktor and OkHttp, which are designed for creating REST APIs, WebSocket servers and making network requests in Kotlin. Another useful library from Kotlin is Kotlin Serialization which allows for serialization and de-serialization of Kotlin objects to and from JSON format. As the Auth server will

be making HTTPS requests to communicate with both the **RADIUS** server and the **IDP**, these libraries and framework are particularly useful.

Here is a list of all the libraries and frameworks used in this project, along with their purposes:

- Core Kotlin and Serialization
  1. Kotlin Standard Library (org.jetbrains.kotlin:kotlin-stdlib) [78]: Kotlin Standard Library.
  2. Kotlinx Serialization JSON (org.jetbrains.kotlinx:kotlinx-serialization-json) [63]: Provides JSON serialization and de-serialization for Kotlin objects.
- Server Framework (Ktor) [77]
  1. Ktor Server Core (io.ktor:ktor-server-core): Contains core Ktor functionality.
  2. Ktor Netty Engine (io.ktor:ktor-server-netty): A dependency for the ktor-server-netty engine.
  3. Ktor Serialization (io.ktor:ktor-serialization): To serialize/deserialize JSON data.
- Database
  1. RocksDB JNI (org.rocksdb:rocksdbjni) [82]: RocksDB fat jar, interface to use RocksDB database.
- Networking (HTTP Client) [81]
  1. OkHttp (com.squareup.okhttp3:okhttp): HTTP client for making network requests from the server.
  2. OkHttp Logging Interceptor (com.squareup.okhttp3:logging-interceptor): Useful for logging HTTP request and response data.
- Firebase Integration [51]
  1. Firebase Admin SDK (com.google.firebase:firebase-admin): Allows to interact with Firebase from privileged environments to perform queries, generate and verify Firebase auth tokens, ...
- Logging
  1. SLF4J API (org.slf4j:slf4j-simple) [83]: Used to unify logging across different libraries (used for Logback).

### 4.5.2 Database selection

In the authentication server, the tokens received from the **IDP** will need to be stored to validate user credentials when the **RADIUS** server communicates with the auth server. Comparing the received tokens with those stored in the backend database is essential to ensure that only authenticated users get access to the network.



## Data format

As the tokens typically expire after some time (e.g., the access tokens usually expire after one hour) and many requests are made to the auth server, the tokens will be stored in a key-value store [103]. A key-value store is a type of NoSQL database (non-relational type of database) that uses pairs composed of a key and a value. The key is a unique identifier used to access the associated value and is usually a string or number that allows for fast look-ups. The value is the data that is associated with the key and this value can be any type of data (e.g., strings, numbers, objects, or binary data), depending on the database and the needs of the project. To retrieve data, each key is a pointer to a specific value.

A key-value store is ideal for this project because it features quick look-ups based on a unique key, such as the ID token, and provides fast access to the stored tokens.

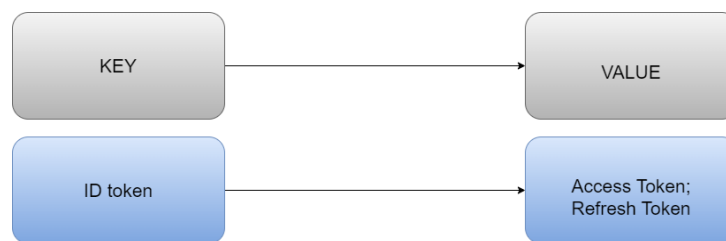


Figure 4.5: Key-value store: concept

For this project, the database will store key-value pairs with the key being the ID token from OIDC and the value will be a concatenation of the Access and Refresh tokens, separated by a semicolon, as represented in Figure 4.5 <sup>3</sup>.

## NoSQL databases

To implement the key-value store, a NoSQL database (non-relational type of database) is needed. More specifically, a key-value database. Key-value database is the simplest type of NoSQL database, but it is extremely powerful. If the data that need to be stored can fit in a key-value database, it is always a good choice to choose it. It is highly efficient, scalable and flexible, and does not require any actual structure for the data. Common use cases for this type of database are user session management, storing user profile information and real-time applications [37].

To choose the right key-value database for this project, one of the first criteria was its popularity. Selecting a widely used database should help ensure that there is good documentation, community support, and resources available. Popular databases typically have a large user base, which means that the challenges or issues encountered during development are more likely to have been addressed by others. Additionally, well-known databases often have robust and secured tools and libraries, which makes integration and maintenance easier.

<sup>3</sup>In addition, a Firebase Cloud Messaging token will eventually be concatenated to the value to allow the auth server to communicate with a specific user. The Figure 4.5 does not show this additional token.

For this first criterion, the "DB-Engines Ranking of Key-value Stores" from [29] and the "Top 26 Key-Value Databases Compared" from [37] were used. Here is some rankings taken from these websites (the score is computed according to the popularity of the database):

Database	Strengths (Dragonfly)	Weaknesses (Dragonfly)	Score (DB-Engines)	# of visits (Dragonfly)
Redis	Fast data access, Rich data structures, High availability, Persistence options	Limited query capabilities, Data size limited by memory	148.64	444.0k
RocksDB	Highly customizable, Support for atomic writes, Compression and compaction for efficient storage	Requires manual management of some operations, Steeper learning curve for advanced features	2.96	25.5k

Table 4.2: Key-Value store database comparison based on [37] [29]

Based on Table 4.2, Redis and RocksDB are both a good option. Note that the Cassandra database was also considered as it supports stores and can be distributed. However, this capability may be an overkill if the application does not require extensive scaling or high-volume data handling [47], which is the case here. Cassandra was therefore not chosen.

Finally, RocksDB [73] [60] was chosen over Redis for this project because it aligns better with the requirements of an embedded database, as the database system will be integrated directly into the auth server application, running within the same process as the application itself. Although Redis is often used as a high-speed, in-memory caching layer with optional persistence, RocksDB is specifically optimized for high-performance embedded projects.

RocksDB can handle heavy read and write workloads directly on disk, making it particularly suitable when data persistence is a primary requirement, even in the cases of application restarts or server failures. Additionally, RocksDB can handle large datasets on a single machine without relying heavily on memory. [30] [110]

### RocksDB and its key features [40]

As already mentioned, RocksDB is an embedded high-performance key-value database designed by Facebook, built specifically to handle large volumes of data and optimize both read and write performance on flash and solid-state drives. It is an evolution of Google LevelDB.

RocksDB uses an LSM tree data structure, which organizes data into different levels on disk. This structure is optimized for fast writes by first storing incoming data in memory and then organizing it efficiently on disk in batches, reducing I/O operations and optimizing for SSDs.

RocksDB is designed to handle high write volumes with minimal impact on performances. Data is first stored in memory and then periodically written to disk.

Common use cases of RocksDB are embedded systems and caching layers.

## 4.6 IDP

### 4.6.1 Authentication method

To verify the identity of a user, the IDP needs to prompt the user for credentials such as usernames and passwords. In this project, the users will use two types of credentials: Google credentials and **e-ID** credentials.

Google credentials were chosen for development because they are very similar to the authentication process used for **e-ID** credentials, which are managed by BOSA (Belgian Government Service) [108]. Since the **e-ID** IDP registration to set up an OAuth 2.0 client id and secret has to be done by BOSA itself, and authorization/agreements are needed, it was necessary to wait for BOSA to set everything up. On the other hand, the Firebase [51]/Google Identity [34] IDP allows you to set up an OAuth 2.0 client id and secret on your own via their website.

By starting with Google OAuth 2.0, which is also OAuth-based, the development process was not stopped while waiting for the credentials from BOSA, with the authentication flow being implemented similarly to how it will work with the **e-ID** system. Once the **e-ID** credentials and configuration are provided by BOSA, the system can switch to **e-ID** by updating the client id, secret, and endpoints in the code, as the underlying OAuth mechanism will remain unchanged. This approach allowed development to move forward without needing to wait for anything and provides a second authentication method for this project.

### 4.6.2 Google credentials: Firebase/Google Identity

Google OAuth is based on the OAuth 2.0 protocol, provided through Google Identity services [34], which allows users to authenticate via their Google account and authorize external applications (such as websites, mobile applications and other services) to access their data (such as user information, Calendar, etc.) without directly sharing their Google account credentials, as can be seen in Figure 4.6.

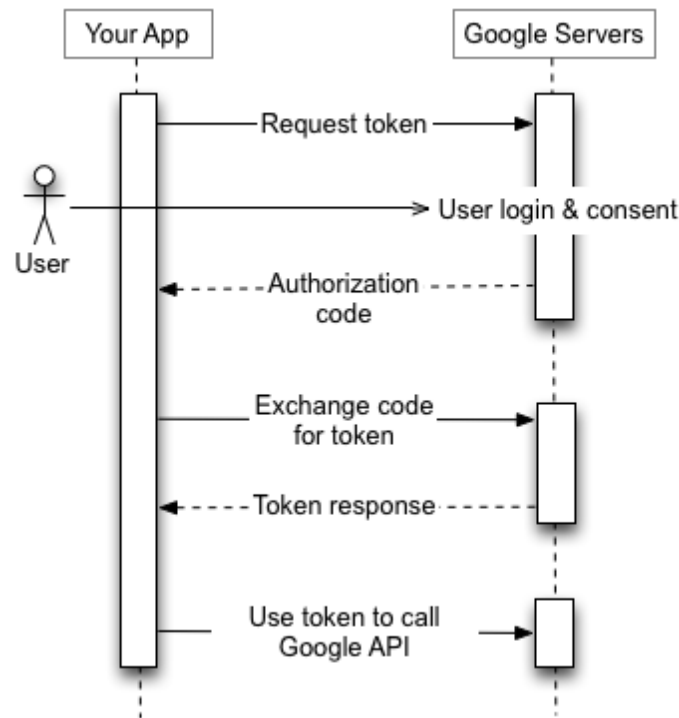


Figure 4.6: The authorization sequence with Google APIs that use the OAuth 2.0 protocol, taken from [34]

Google is an authentication method supported by Firebase [51]/Google Identity [34]. By using Firebase/Google Identity as the IDP, authentication and user management are efficiently handled, as Firebase simplifies the integration of the Google authentication system. The Google Identity service [34] allows users to authenticate through their Google accounts and issue tokens once the user is successfully authenticated, while Firebase provides tools that simplify the implementation of authentication in the application and abstracts much of the complexity involved in managing user sessions, handling authentication flows, and interacting with the Google Identity Platform. It is thus necessary to set up both the Firebase and Google identity for the API. This process will be described in the chapter 5.

### Authorize Request

The authorization code request format is described in the "Using OAuth 2.0 for Web Server Applications" page from the Google Identity documentation [34]. The request should be directed to the Google authorization code endpoint (<https://accounts.google.com/o/oauth2/v2/auth>) with, for example, the following parameters:

- `client_id`: mandatory, the client ID mentioned on the API Console Credentials page.
- `redirect_uri`: mandatory, the redirect URIs mentioned in the API Console Credentials page for the given `client_id`.
- `response_type`: mandatory, must contain the response type code [9].
- `scope`: mandatory, must contain at least the `openid` scope. For this application, the profile scope can also be used to retrieve user information and personalize the

application.

Other parameters can be found in the documentation. Here is an example taken from the documentation of a request for authorization code:

```
https://accounts.google.com/o/oauth2/v2/auth?
scope=https%3A//www.googleapis.com/auth/drive.metadata.readonly%20https%3A/
/www.googleapis.com/auth/calendar.readonly&
access_type=offline&
include_granted_scopes=true&
response_type=code&
state=state_parameter_passthrough_value&
redirect_uri=https%3A//oauth2.example.com/code&
client_id=client_id
```

After this request, the user will be prompted to accept or refuse to grant access to the application.

The OAuth 2.0 server will then respond to the application via the redirect URL specified before. The authorization code will be on the query string:

```
https://oauth2.example.com/auth?code=4/P7q7W91a-oMsCeLvIaQm6bTrgtp7
```

The authorization code in this example is thus *4/P7q7W91a – oMsCeLvIaQm6bTrgtp7*.

### Token Request

The HTTP POST request to exchange the authorization code for refresh and access tokens must have the same format as specified on the "Using OAuth 2.0 for Web Server Applications" page from the Google Identity documentation [34]. The *https : //oauth2.googleapis.com/token* endpoint is called with the following parameters:

- `client_id`: mandatory, the client ID mentioned on the API Console Credentials page [5].
- `client_secret`: mandatory, the client secret also mentioned on the API Console Credentials page.
- `code`: mandatory, the authorization code.
- `grant_type`: mandatory, must be "authorization\_code".
- `redirect_uri`: mandatory, the redirect URIs mentioned in the API Console Credentials page for the given `client_id`.

Here is an example of request for tokens taken from the Google documentation:

```
POST /token HTTP/1.1
Host: oauth2.googleapis.com
Content-Type: application/x-www-form-urlencoded
```

```
code=4/P7q7W91a-oMsCeLvIaQm6bTrgtp7&
client_id=your_client_id&
client_secret=your_client_secret&
redirect_uri=https%3A//oauth2.example.com/code&
grant_type=authorization_code
```

In response, the Google server will send a JSON response that will contain the following attributes:

- `access_token`
- `expires_in`: the time of validity of the tokens received.
- `refresh_token`
- `scope`: the requested scopes.
- `token_type`: the type of token, always set to Bearer.

A typical response, taken from the Google documentation, looks like this:

```
{
  "access_token": "1/fFAGRNJru1FTz70BzhT3Zg",
  "expires_in": 3920,
  "token_type": "Bearer",
  "scope": "https://www.googleapis.com/auth/drive.metadata.readonly
           https://www.googleapis.com/auth/calendar.readonly",
  "refresh_token": "1//xEoDL4iW3cx1I7yDbSRFYNG01kVKM2C-259HOF2aQbI"
}
```

### 4.6.3 e-ID: FOD BOSA's FAS

e-ID is an authentication method supported by the FPS Policy and Support Federal Authentication Service (BOSA FAS) [44] [108]. The OIDC authorization sequence from BOSA is represented in Figure 4.7

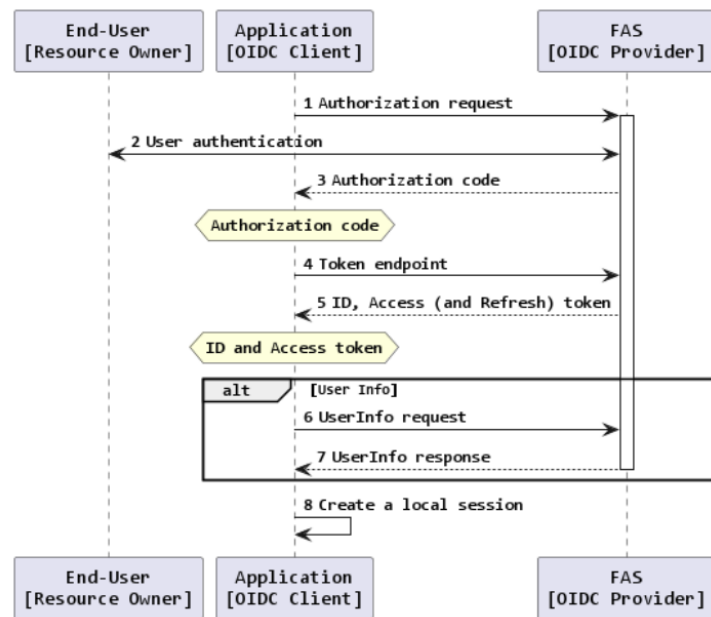


Figure 4.7: The authorization sequence with BOSA APIs that use the OAuth 2.0 protocol, taken from [44]

As already mentioned, the sequence for Google authentication and e-ID authentication both use OIDC and are thus really similar.

The BOSA documentation [44] mentions every OpenID Connect endpoint URL needed to contact the provider, the scopes and claims and the Level of Assurance in the authentication context.

## Authorize Request

The authorization code request format is also described in the document. To obtain the initial authorization code, an HTTP GET request needs to be made to the Authorization Code endpoint of the FAS (.../fas/oauth2/authorize) with, for example, the following parameters:

- `scope`: mandatory, must contain at least the `openid` scope. For this application, the `profile` scope can also be used to retrieve user information and personalize the application.
- `response_type`: mandatory, must contain the response type code [9].
- `client_id`: optional.
- `redirect_uri`: optional, the redirect URI mentioned during the on-boarding of the client with BOSA.
- `state`: used to maintain state between the request and the response.

Other parameters can be found in the documentation. Here is an example taken from the documentation of a request for authorization code using BOSA:

```
GET https://idp.iamfas.int.belgium.be/fas/oauth2/authorize
?response_type=code
```

```

&client_id=myclientid
&scope=openid%20profile
&acr_values=urn:be:fedict:iam:fas:Level500
&redirect_uri=https://www.google.com
&state=af0ifjsldkj
&nonce=1244542

```

In response to this request, an HTTP GET request will be made to the `redirect_uri` mentioned in the request. It will contain the following parameters:

- `scope`: the requested scopes.
- `code`: the authorization response code.
- `state`: to maintain state between the request and the response.
- `client_id`
- `is`: the issuer of the authorization code.

Here is an example taken from the documentation of a response for authorization code using BOSA:

```

GET redirect_uri (http(s)://...)
?code=31308323-c08e-431a-a5dc-2e7335795b43
&scope=openid%20profile
&iss=https%3A%2F%2Fidp.iamfas.int.belgium.be%2Ffas%2Foauth2
&state=af0ifjsldkj
&client_id=myclientid

```

## Token Request

Once the authorization code is received, the client can then use it to request the id, access and refresh tokens. The request is a HTTP POST to the token endpoint of the FAS (`.../fas/oauth2/access_token`). It contains the following headers:

- `Authorization`: mandatory, contains the `client_id` and its secret.
- `Content-Type`: mandatory, must be `application JSON` for this project.

It must also contain the following parameters:

- `grant_type`: mandatory, must be `"authorization_code"`.
- `code`: mandatory, the actual authorization code.
- `redirect_uri`: mandatory, the redirect URI mentioned during the on-boarding of the client with BOSA.

Here is an example of request for tokens taken from the documentation:

```

POST https://idp.iamfas.int.belgium.be/fas/oauth2/access_token
?grant_type=authorization_code
&code=HusR0KJVsnVHJ84myuMn5taiqi4

```



```
&redirect_uri=https://redirecturi.be
headers: Authorization: Basic v2xGZW50aWN6Y2xpZW50c2VjcmV0
Content-Type: application/x-www-form-urlencoded
```

In response, the server will send a response that will contain the following attributes:

- scope: the requested scopes.
- access\_token
- refresh\_token
- token\_type: the type of token, for example Bearer.
- expires\_in: the time of validity of the tokens received.
- id\_token

A typical response taken from the documentation looks like this:

[illegible]

Also note that the ID token is a JWT.

## Google and e-ID

Given the schema for the authorization sequence that uses the OAuth 2.0 protocol, the authorization request format and token request format, it is now clear that using Google or e-ID credentials is really similar. Using one method or the other will only require minimal adjustments and Google authentication is thus used first for development purposes.

#### 4.6.4 IDP on-boarding

For this project, the IDP will need to become a member of the **OpenRoaming** federation via RADSEC using the Cisco Intermediate CA (WBA root) as described in the Cisco documentation [16].

To become a member, a few steps need to be taken:

- The EAP/RADIUS server needs to be available on the Internet. A Certificate Signing Request (CSR) will need to be completed in order for the Cisco Intermediate CA to sign

the certificate. As already mentioned, the server for this project is *marie.tiedie.io*. The realm chosen that is included in the CSR is *test – beid.openroaming.net*.

- The certificate that will be received after being signed needs to be installed on the server.
- DNS needs to be configured to ensure that the realms are discoverable.

## DNS records

To ensure that the IDP is discoverable by the OpenRoaming federation and its members, 3 DNS records must be created for the realm in the DNS that is authoritative for the domain *test – beid.openroaming.net*. This domain is a subdomain of *beid.openroaming.net* for the IDP set up of this project. The 3 records are:

- The NAPTR record for the realm: *\_radsec.\_tcp.test-beid.openroaming.net*. 300 IN SRV 0 10 2083 *marie.tiedie.io*.
- The SRV record for the pointer: *test-beid.openroaming.net*. 300 IN NAPTR 50 50 "s" "aaa+auth:radius.tls.tcp" "" *\_radsec.\_tcp.test-beid.openroaming.net*.
- The address record for the RADSEC endpoint: *marie.tiedie.io* pointing to 185.48.12.253.

## 4.7 Final solution

Figure 4.8 summarizes every component together with the technologies and software used for this project.

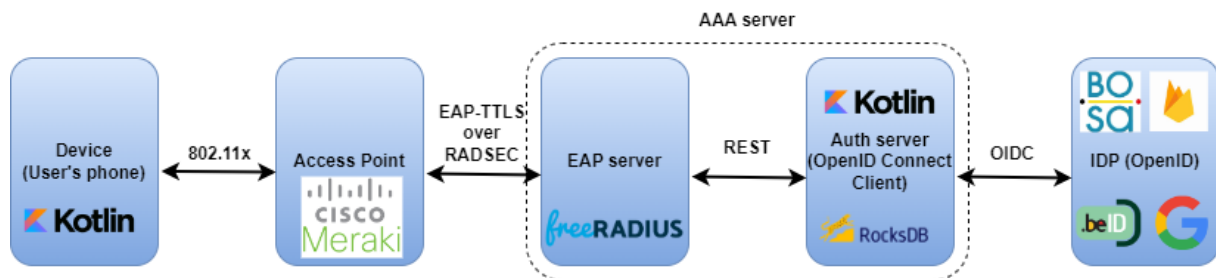


Figure 4.8: OpenRoaming: Components of the project and associated technologies and softwares

The mobile device coded with Kotlin seamlessly connects to a Wi-Fi network using the Hotspot 2.0 profile that contains its credentials and initiates authentication using the 802.1X protocol for secure access.

The Meraki access point receives the device connection request and forwards the user credentials securely using the EAP-TTLS over RADSEC tunnel to the EAP/RADIUS server. This allows encrypted communication between the device and the server.

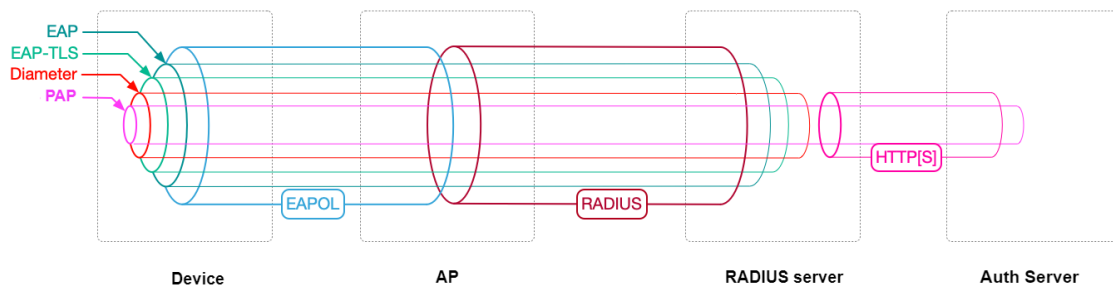
FreeRADIUS, acting as the EAP/RADIUS server, manages the authentication process by verifying the credentials. It communicates with the Auth Server via REST to confirm the user's identity.

The auth server, coded in Kotlin, serves as an OpenID Connect client. It uses RocksDB, the key-value database, to store and manage the tokens it received from the IDP. It is

responsible for receiving requests coming from the **EAP/RADIUS** server and checking if the credentials provided are valid according to its database.

Figure 4.9 summarize the flow of authentication protocols in a setup in which a device connects to a network using **EAP-TTLS** with plaintext authentication (**PAP**) and authenticates against a REST API through the auth server.

#### EAP-TTLS with plaintext auth against REST API



Author: Arsen Cudbard-Bell  
Copyright 2018 The FreeRADIUS project  
Creative Commons license CC BY

Figure 4.9: Final solution: EAP-TTLS with PAP over RADSEC taken from [27] and slightly modified to suit this project

Finally, with this setup and for the user to seamlessly connect to an **OpenRoaming**-enabled network, the user must previously authenticate with the **IDP** to gain network access. When successfully authenticated, the **IDP** will provide ID, access and refresh tokens to the auth server that will put them in a profile for the user. The user device will then use these tokens as credentials to connect to the network while the auth server will verify the credentials received from the **RADIUS** server when a device tries to connect to the network.

# Chapter 5

## Prototype: implementation and demonstration

### 5.1 Source code and other resources

The implementation of the prototype and various other resources are hosted in a gitlab directory:

<https://gitlab.uliege.be/Marie.Maes/openroaming>

The gitlab directory is organized as follows:

## CHAPTER 5. PROTOTYPE: IMPLEMENTATION AND DEMONSTRATION

```
openroaming/
├── freeradius/3.0/
│   ├── users
│   └── certs (not shown in gitlab)
│       ├── clients.conf
│       ├── sites-enabled/
│       │   ├── default
│       │   ├── inner-tunnel
│       │   └── tls
│       ├── mods-enabled/
│       │   ├── eap
│       │   └── rest
│       ├── ...
│       └── README.md
├── auth-server/
│   ├── build.gradle.kts
│   ├── src/main/kotlin/com/exemple/
│   │   └── Application.kt
│   ├── ...
│   └── README.md
├── client-app/
│   ├── app/
│   │   ├── src/main/java/com/exemple/firebaseauth
│   │   │   ├── auth/
│   │   │   ├── navigation/
│   │   │   ├── ui/
│   │   │   ├── utils/
│   │   │   └── MainActivity.kt
│   │   ├── build.gradle.kts
│   │   └── google-services.json
│   ├── build.gradle.kts
│   └── README.md
├── latex/
│   ├── OpenRoaming.pdf
│   └── OpenRoaming.tex
└── README.md
```

## 5.2 Device (user's phone)

The mobile application that will be installed on the user's device must allow the user to authenticate with the IDP using either Google credentials or e-ID credentials. When authenticated successfully, it must be able to download the **OpenRoaming** profile that will allow the device to seamlessly connect to the **OpenRoaming**-enabled networks. Some of the main classes and functions to achieve this are described below. The Figure 5.3 shows some screens of the mobile application.

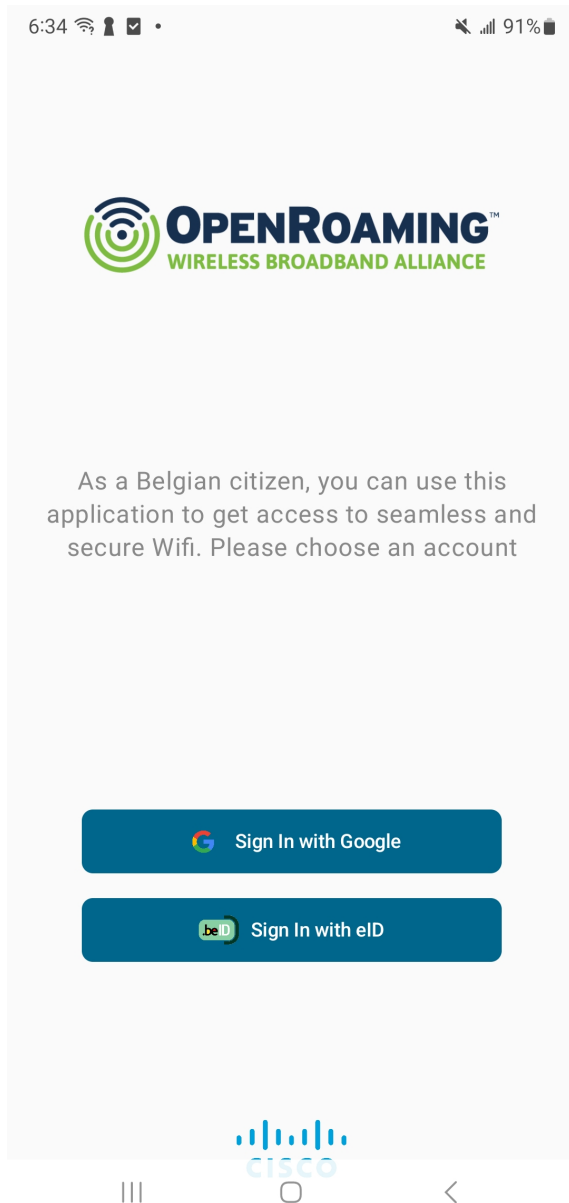


Figure 5.1: User interface for authentication with two buttons, one for Google and the other for e-ID

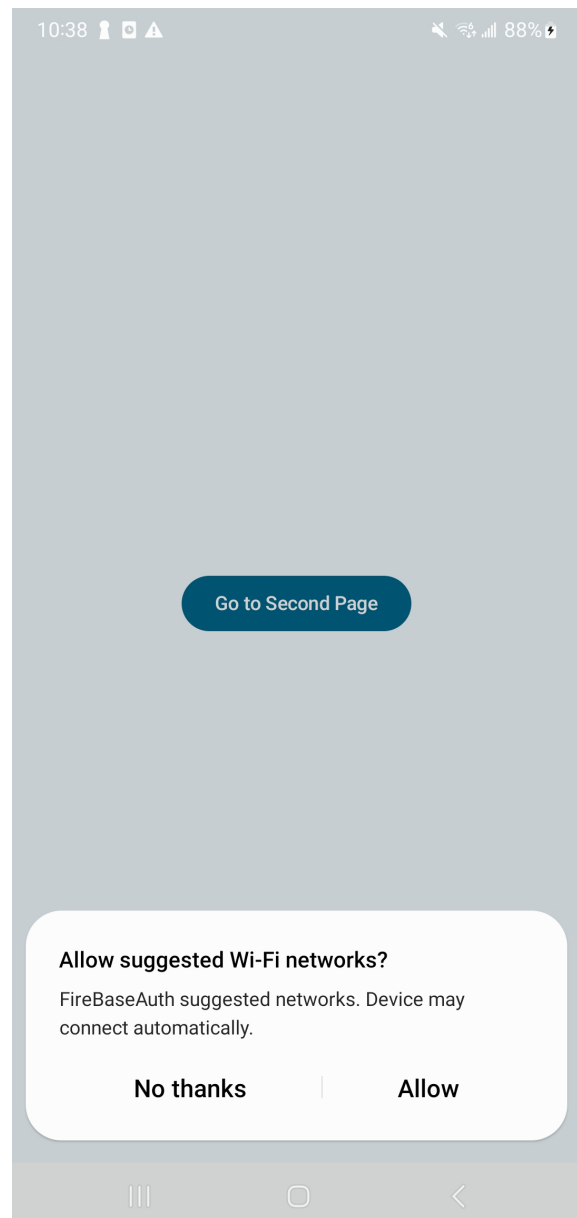


Figure 5.2: Wifi suggestion pop-up to add the Passpoint profile

Figure 5.3: Screenshots of the mobile application

### 5.2.1 MainActivity

The *MainActivity* class is the class that sets up the navigation, themes, authentication logic, and everything necessary for the application to work properly.

First, an instance of *AuthViewModel* is created using the *AuthViewModelFactory* factory that uses the *AuthRepository*. This view model is used to manage the authentication state of the application and allows to know if the user is either logged in or not. Then, the *rememberNavController* function is used to create a navigation controller that manages the transitions between the different screens.

The authentication state is known through the *authState.collectAsState* function of the *AuthViewModel*. The function checks for any changes in the user's authentication status. Based on this state, the application determines the screen with which the user will start. If the user is authenticated, the application navigates to the main content page (main). If the user is not authenticated yet, the login screen (auth) is selected.

Finally, the *NavGraph* function is called. This function defines the navigation structure of the application, linking the navigation controller and the view model to the different screens.

The *NavGraph* defines how the user can navigate between the two screens of the application. These two screens are the authentication screen and the main screen, which is a welcome page for authenticated users and will allow the user to sign out.

### 5.2.2 AuthScreen

When the user is not yet authenticated, he is redirected to the *AuthScreen* function. The *AuthScreen* is used as the user interface for authentication, as can be seen on Figure 5.1. It prompts unauthenticated users to sign in via Google or e-ID via a button and, when the button is pushed, it first verifies if the device is connected to the Internet before trying any authentication process. It displays a toast message if there is no internet connection.

If the Google button is pushed, the function then launches the Google sign-in flow using the *GoogleSignIn* function. It then retrieves an ID token and an authorization code from Google upon successful sign-in and sends the authorization code to the auth server to exchange it for access and refresh tokens via the *sendAuthCodeToServer* function. The auth server will then respond with the id, access and refresh tokens.

Finally, the *AuthScreen* function hashes the access token and sends a request to the auth server for the Hotspot 2.0 profile for seamless Wi-Fi access via the *sendGenerateAndroidProfileRequest* function.

For now, the e-ID Sign-In button is a placeholder, as it provides a button for e-ID login but the functionality is not implemented in this screen yet.

### 5.2.3 AuthUtils

Both the *sendAuthCodeToServer* and *sendGenerateAndroidProfileRequest* functions are defined in *AuthUtils.kt*. Indeed, the *AuthUtils* class contains utility functions and data structures to handle communication between the mobile application and the auth server.

First, the *getOkHttpClient* function is used to configure an *OkHttpClient* with a custom CA certificate for secure communication with the auth server over HTTPS. The CA certificate is the one who signed the auth server certificate. It is necessary to provide this CA certificate in order for the device to trust the backend server.

The *sendAuthCodeToServer* function sends the Google authorization code (*authCode*) and the Firebase Cloud Messaging (FCM) [50] token to the auth server. The FCM token is a token that uniquely identifies the client application and is required for the auth server to be able to reliably send messages to the client application. This token is retrieved using the *getFCMToken* function.

The message is sent to the auth server through the *https : //marie.tiedie.io/auth* URL. It constructs a JSON body containing *authCode* and *fcmToken* and sends it as a POST request. The server will then respond with the *idToken* and *accessToken*.

Finally, the *sendGenerateAndroidProfileRequest* function sends a request to the auth server to generate an Android Passpoint (Wi-Fi) profile via the *https : //marie.tiedie.io/generateAndroidProfile* URL. It sends a GET request with query parameters (*friendlyName*, *username*, *password*) needed in the profile. It then parses the Passpoint configuration from the server response and uses the *WifiManager* [54] to add a Wi-Fi suggestion [52] for the device. As can be seen in Figure 5.2, it triggers a pop-up that the user needs to allow the profile to be installed.

### 5.2.4 FMS

The *MyFirebaseMessagingService* is a custom implementation of the Firebase Messaging Service used to handle push notifications and messages sent from Firebase Cloud Messaging (FCM). This service is responsible for processing incoming messages, including both data and notification payloads. The FCM is only used when the auth server refreshes the access token and thus needs to notify the client that its current profile (which contains the access token) is not up to date.

Upon receiving a message from the auth server, it knows that the access token is updated, and thus extracts the data payload which is the *accessToken* and *idToken*. The *accessToken* is hashed using SHA-256 and then Base64-encoded for security purposes. An *AndroidProfileRequest* object is then created and a new profile request is sent to the auth server. To do so, it simply calls the *sendGenerateAndroidProfileRequest* function to send the generated profile to the server and configure the Passpoint profile of the device.



### 5.3 Access point

The AP must be configured to leverage **RADIUS** authentication with WPA2-Enterprise [19]. Thus, it will work using **RADSEC** [18].

To configure that, the Meraki dashboard allows one to configure access control for an SSID on the Wireless > Configure > Access control page, as can be seen in Figure 5.6. Note that the Meraki AP is reachable using the IP address 81.245.174.125. The SSID configured here is Test1 and in this project, WPA2 Enterprise will be used with a **RADIUS** server. The user credentials are validated with 802.1X at the association time, as mentioned in Figure 5.4.

The **RADIUS** server *marie.tiedie.io* is added in the **RADIUS** section, as shown in Figure 5.5. Its FQDN is the domain name of the server (which has IP 185.48.12.253). The server Auth port is 2083, which is the default port for **RADSEC**. The secret is by default **RADSEC** and also needs to be configured on the FreeRADIUS server to make it work. **RADSEC** needs to be enabled.

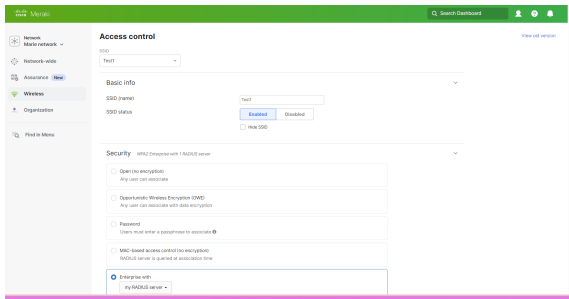


Figure 5.4: SSID Test1 configuration page for access control

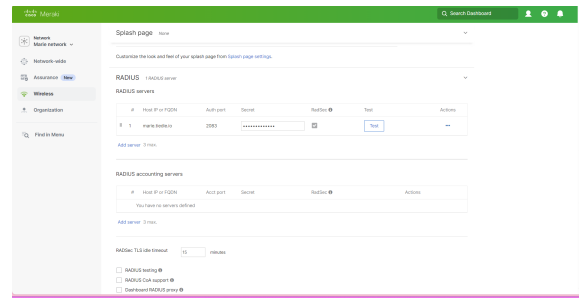


Figure 5.5: SSID Test1 configuration page for access control: **RADIUS** server configuration

Figure 5.6: Meraki dashboard overview [20]

When everything is set up to work with **RADIUS** authentication and WPA2-Enterprise, a **TLS** tunnel will thus be established using certificates between the AP and the **RADIUS** server, as can be seen in Figure 5.7.

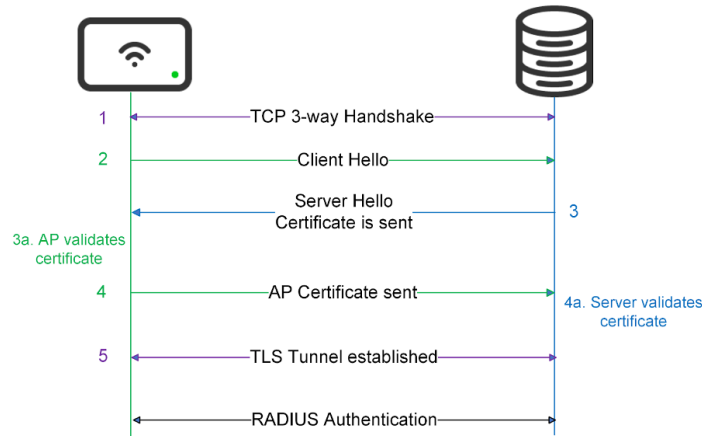
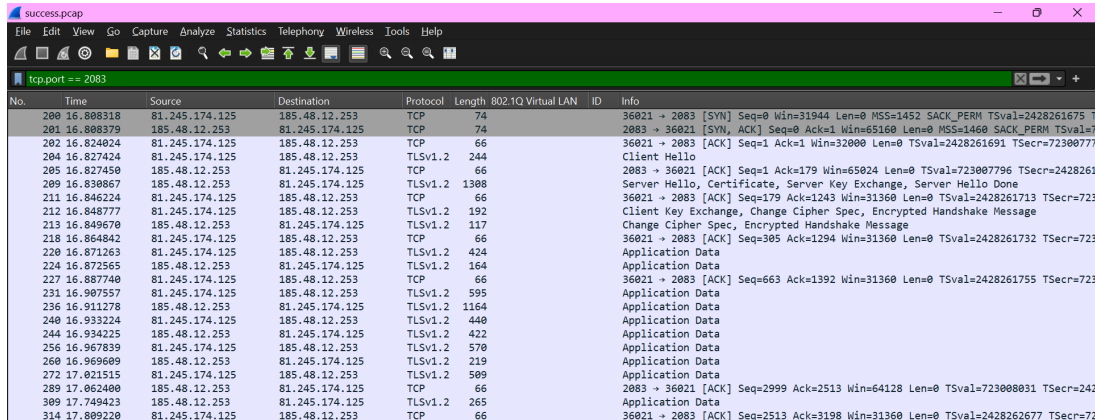


Figure 5.7: The **TLS** Tunnel establishment, taken from [18]

## CHAPTER 5. PROTOTYPE: IMPLEMENTATION AND DEMONSTRATION

When a client tries to connect to the network via the access point, the packets exchanged between the AP and the RADIUS server can be captured using tcpdump and filtering on the tcp port 2083 (RADSEC default port) and look like this:



No.	Time	Source	Destination	Protocol	Length	802.1Q	Virtual LAN	ID	Info
200	16.808318	81.245.174.125	185.48.12.253	TCP	74				36021 → 2083 [SYN] Seq=0 Win=31344 Len=0 MSS=1452 SACK_PERM TSval=2428261675 TSecr=723080777
201	16.808379	185.48.12.253	81.245.174.125	TCP	74				2083 → 36021 [SYN, ACK] Seq=0 Ack=1 Win=5160 Len=0 MSS=1460 SACK_PERM TSval=723080777 TSecr=0
202	16.824024	81.245.174.125	185.48.12.253	TCP	66				36021 → 2083 [ACK] Seq=1 Ack=1 Win=32000 Len=0 TSval=2428261691 TSecr=723080777
204	16.827424	81.245.174.125	185.48.12.253	TLSv1.2	244				Client Hello
205	16.827450	185.48.12.253	81.245.174.125	TCP	66				2083 → 36021 [ACK] Seq=1 Ack=179 Win=65024 Len=0 TSval=723080779 TSecr=2428261
209	16.830867	185.48.12.253	81.245.174.125	TLSv1.2	1308				Server Hello, Certificate, Server Key Exchange, Server Hello Done
211	16.846224	81.245.174.125	185.48.12.253	TCP	66				36021 → 2083 [ACK] Seq=179 Ack=1243 Win=31360 Len=0 TSval=2428261713 TSecr=723
212	16.846777	81.245.174.125	185.48.12.253	TLSv1.2	192				Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
213	16.849670	185.48.12.253	81.245.174.125	TLSv1.2	117				Change Cipher Spec, Encrypted Handshake Message
218	16.864842	81.245.174.125	185.48.12.253	TCP	66				36021 → 2083 [ACK] Seq=305 Ack=1294 Win=31360 Len=0 TSval=2428261732 TSecr=723
220	16.871263	81.245.174.125	185.48.12.253	TLSv1.2	424				Application Data
224	16.872565	185.48.12.253	81.245.174.125	TLSv1.2	164				Application Data
227	16.887740	81.245.174.125	185.48.12.253	TCP	66				36021 → 2083 [ACK] Seq=663 Ack=1392 Win=31360 Len=0 TSval=2428261755 TSecr=723
231	16.907557	81.245.174.125	185.48.12.253	TLSv1.2	595				Application Data
236	16.911278	185.48.12.253	81.245.174.125	TLSv1.2	1164				Application Data
240	16.933224	81.245.174.125	185.48.12.253	TLSv1.2	440				Application Data
244	16.934225	185.48.12.253	81.245.174.125	TLSv1.2	422				Application Data
256	16.967839	81.245.174.125	185.48.12.253	TLSv1.2	570				Application Data
260	16.969609	185.48.12.253	81.245.174.125	TLSv1.2	219				Application Data
272	17.021515	81.245.174.125	185.48.12.253	TLSv1.2	509				Application Data
285	17.062400	185.48.12.253	81.245.174.125	TCP	66				2083 → 36021 [ACK] Seq=2999 Ack=2513 Win=64128 Len=0 TSval=7230808031 TSecr=242
309	17.749423	185.48.12.253	81.245.174.125	TLSv1.2	265				Application Data
314	17.809220	81.245.174.125	185.48.12.253	TCP	66				36021 → 2083 [ACK] Seq=2513 Ack=3198 Win=31360 Len=0 TSval=2428262677 TSecr=72

Figure 5.8: The TLS Tunnel establishment between the Meraki AP and the RADIUS server

The first three packets correspond to the TCP 3-way handshake, followed but the Client Hello, Server Hello, Certificate, etc., as mentioned in the Figure 5.7.

### 5.4 OpenRoaming considerations: Cisco Spaces

To integrate OpenRoaming with Cisco Spaces, it is necessary to create an OpenRoaming profile, enabling hotspot on connectors, and configuring controllers to associate them with the profiles. To do so, the documentation "OpenRoaming integration with Cisco Spaces" [17] was used. The goal is to link Cisco Spaces to the Meraki Cloud and then to configure an OpenRoaming-enabled SSID. Its access policy will be set to either "Accept all authenticated users" or "Accept only your users" and preferred credentials will be specified to favor the e-ID credentials, as can be seen in Figure 5.9.

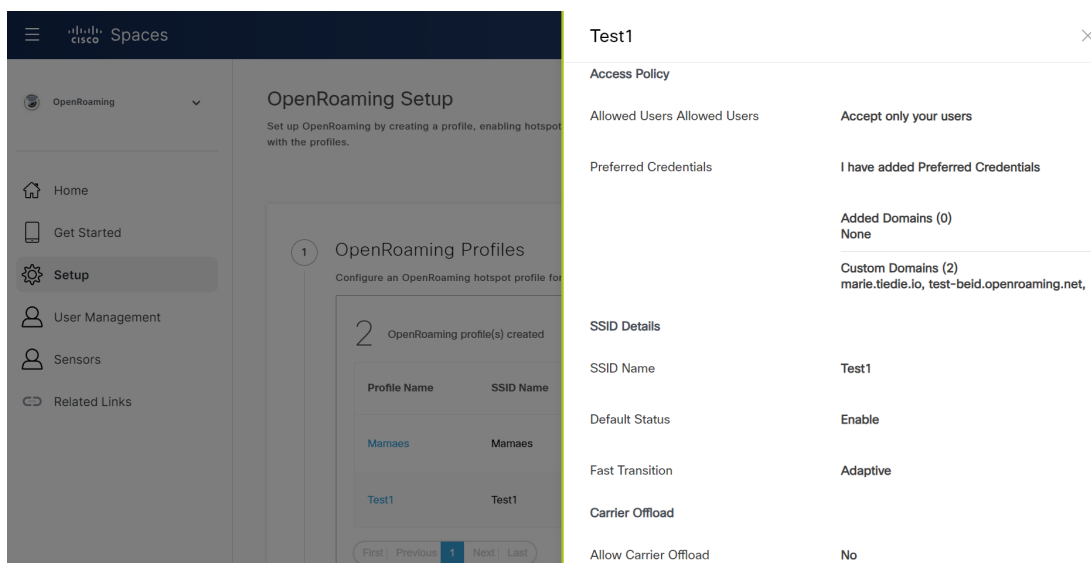


Figure 5.9: OpenRoaming Setup of the Test1 SSID in Cisco Spaces

After this setup, the Meraki configuration will be slightly modified by this integration, as the devices will now send **RADIUS** Access-Request packets via a Meraki proxy, which will forward these messages to the specified **RADIUS** servers.

## 5.5 EAP/RADIUS server

The goal of the EAP server is to set up REST and an EAP-TTLS tunnel with PAP over RADSEC (**RADIUS** over **TLS**) on FreeRADIUS. FreeRADIUS provides strong documentation for configuration as it can be complex [114].

Note that the server is *marie.tiedie.io* (185.48.12.253) and ssh is needed to access it. To run the server in debug mode, this command is used:

```
freeradius -fxx -l stdout
```

For testing purposes, a local user account can be added in */etc/freeradius/3.0/users* for authentication:

```
testuser Cleartext-Password := "password"
```

This allows any device to get access to the network by connecting using the right SSID in the Wi-Fi settings and entering the user "testuser" with the password "password" to authenticate. The user credentials are stored in plain text for testing purposes.

When these configurations are done, the FreeRADIUS server should thus be able to use:

- EAP-TTLS with PAP for Wi-Fi authentication request coming from the AP.
- RADSEC (**RADIUS** over **TLS**) for secure **RADIUS** traffic.
- REST to integrate the auth server that will validate the credentials received.

### 5.5.1 Certificates generation

To be able to use RADSEC (**RADIUS** over **TLS**), a server certificate is required for the TLS handshake. This certificate is signed by the Cisco CA and will enable the access point to establish a secure TLS tunnel with the server.

In addition to the certificate used in the TLS handshake, a certificate is needed for the EAP-TTLS conversation. OpenSSL is used for this purpose.

First, the goal is to create a Certificate Authority (CA):

```
openssl genrsa -out ca.key 2048
openssl req -x509 -new -nodes -key ca.key -sha256 -days 1024 -out ca.pem
```

This creates a CA key (ca.key) and a CA certificate (ca.pem) which will sign the server certificate. The first command generates a 2048-bit RSA private key for the CA while the second command creates a self-signed certificate using the CA private key (ca.key) and valid for 1024 days, stored in ca.pem.

Then, the server certificate that is signed with the CA is created:

```
openssl genrsa -out server.key 2048
openssl req -new -key server.key -out server.csr (-config config.cnf)
openssl x509 -req -in server.csr -CA ca.pem -CAkey ca.key -CAcreateserial
-out eap_server.pem -days 500 -sha256
```

This creates the server's private key (server.key) and a signed server certificate (server.pem), required for EAP-TTLS. The first command generates the private key for the server. The second command then creates a Certificate Signing Request (CSR) for the server. Finally, the last command is used to sign the server's CSR with the CA certificate and key, creating the server's certificate (server.pem) with a validity of 500 days.

All the generated files are placed in the "certs" folder of FreeRADIUS. Also note that the self-signed CA will be placed in the Meraki cloud controller via its dashboard to allow the access point to trust this CA.

These commands ensure that the FreeRADIUS server has the correct permissions to access the server key file and then restart the server to ensure the permissions are set:

```
sudo chmod 640 /etc/freeradius/3.0/certs/server.key
sudo chown freerad:freerad /etc/freeradius/3.0/certs/server.key
sudo systemctl restart freeradius
```

These commands restrict permissions on server.key, allowing only the FreeRADIUS server to access it and then set the file owner and group to FreeRADIUS's system user.

### 5.5.2 RADSEC configuration

To add a RADSEC client, it is necessary to edit */etc/freeradius/3.0/clients.conf*:

```
1  client radsec_client {
2      ipaddr = <Meraki AP IP> (81.245.174.125)
3      secret = radsec
4      proto = tcp
5      tls {
6          ca_file = /etc/freeradius/3.0/certs/ca.pem
7          certificate_file = /etc/freeradius/3.0/certs/
            radsec_server.pem
```

```

8         private_key_file = /etc/freeradius/3.0/certs/
          radsec_server.key
9     }
10 }

```

Listing 5.1: FreeRadius configuration: RADSEC client

This configuration defines a RADSEC client with the AP's IP (81.245.174.125) and a shared secret for RADSEC authentication. This section also configures the server to use TLS by specifying the paths to the certificate and key files.

Then it is necessary to enable RADSEC at the */etc/freeradius/3.0/sites-available/tls* site:

```

1  listen {
2      ipaddr = 185.48.12.253
3      port = 2083
4      type = auth
5
6      proto = tcp
7
8      # Send packets to the inner tunnel virtual server
9      virtual_server = inner-tunnel
10
11     clients = radsec
12
13     ...
14     tls {
15         #private_key_password = whatever
16         private_key_file = /etc/freeradius/3.0/certs/
          radsec_server.key
17         certificate_file = /etc/freeradius/3.0/certs/
          radsec_server.pem
18         auto_chain = no
19     }
20 }
21
22 clients radsec {
23     client mamaes {
24         ipaddr = 81.245.174.125
25         proto = tls
26         virtual_server = inner-tunnel
27         secret = radsec
28     }
29 }

```

Listing 5.2: FreeRadius configuration: RADSEC configuration

It defines the listening server for incoming RADSEC connections on 185.48.12.253:2083, specifying the IP, port, and protocol type (TCP). This routes requests to the inner-tunnel virtual server to handle inner EAP processing.

### 5.5.3 EAP module

This section explains how EAP can be enabled in FreeRADIUS using the `rlm_eap` module [115].

A module can be enabled (in order for FreeRADIUS to load it) by linking it from the `mods-available` folders to the `mods-enabled` folder:

```
sudo ln -s /etc/freeradius/3.0/mods-available/module
/etc/freeradius/3.0/mods-enabled/
```

It is necessary to edit `/etc/freeradius/3.0/mods-enabled/eap` to make it work with EAP-TTLS [116] [19]:

```

1  eap {
2      default_eap_type = ttls
3      timer_expire = 60
4      ignore_unknown_eap_types = no
5      cisco_accounting_username_bug = no
6
7      tls-config tls-common {
8          #private_key_password = whatever
9          private_key_file = /etc/freeradius/3.0/certs/eap_server
10         .key
11         certificate_file = /etc/freeradius/3.0/certs/eap_server
12         .pem
13     }
14
15     ttls {
16         tls = tls-common
17         default_eap_type = pap
18         copy_request_to_tunnel = yes
19         use_tunneled_reply = yes
20         virtual_server = "inner-tunnel"
21     }
22 }
```

Listing 5.3: FreeRadius configuration: EAP module/EAP-TTLS

It sets **EAP-TTLS** as the default **EAP** type, establishing the secure tunnel. PAP is used within the tunnel as it sends credentials in plaintext, suitable within this secure **TLS** tunnel.

Then, to ensure that PAP is used, */etc/freeradius/3.0/sites-enabled/inner-tunnel* must be edited:

```
1  authorize {
2      eap {
3          ok = return
4      }
5      files
6  }
7  authenticate {
8      Auth-Type PAP {
9          pap
10     }
11     Auth-Type EAP {
12         eap
13     }
14 }
```

Listing 5.4: FreeRadius configuration: PAP and EAP configuration

This setup ensures that PAP is used for authentication within the inner-tunnel, where PAP credentials are securely tunneled.

### 5.5.4 REST module

This section explains how to enable REST in */etc/freeradius/3.0/mods-available/rest* using the `rlm_rest` module [117] [68]. This module is used to translate **RADIUS** authentication requests into HTTP requests and send them to the auth server.

```
1  rest {
2      tls {
3          ca_file = /etc/freeradius/3.0/certs/ca_auth_server.pem
4          ca_path = /etc/freeradius/3.0/certs
5      }
6      connect_uri = "https://marie.tiedie.io:443/"
7
8      authenticate {
9          uri = "${..connect_uri}authenticate"
```

```

10     method = "post"
11     header = 'Content-Type: application/json'
12     body = "json"
13     data = '{"idToken": "%{User-Name}", "accessToken": "%{User-
14           Password}"}'
15     tls = ${..tls}

```

Listing 5.5: FreeRadius configuration: REST module

This configuration enables the REST module to authenticate users by sending the idToken and the accessToken in JSON format to `marie.tiedie.io`. The endpoint where the request is sent is `https://marie.tiedie.io:443/authenticate`.

Finally, the inner tunnel is modified:

```

1     authenticate {
2         Auth-Type REST {
3             rest
4         }
5     }

```

Listing 5.6: FreeRadius configuration: REST configuration

This adds the REST authentication type, allowing FreeRADIUS to use the REST module when authenticating via an external server.

## 5.6 Auth server

The auth server has an embedded database that allows one to manage users and their credentials. It handles incoming requests from users that want to gain access to a network and thus need their credentials to be verified, and stores user tokens when they authenticate to the IDP.

### 5.6.1 Endpoints

The auth server defines 3 main endpoints:

#### POST /auth

It is used to manage interactions between the IDP and the auth server. The /auth endpoint exchanges an authorization code received in a JSON input containing the authCode for tokens (ID token, access token, refresh token) using Google OAuth via the token endpoint of Google. It then stores the tokens in RocksDB, using the idToken as the key and concatenating the accessToken and refreshToken as value. The possible responses from



the server are "200 OK" if authentication succeeds, "500 Internal Server Error" if token exchange or storage fails, and "415 Unsupported Media Type" if the content type is not JSON. To exchange an authorization code for tokens, the *exchangeAuthCodeForTokens* function is called. It constructs an HTTP POST request to the OAuth provider's token endpoint. The request must contain the required parameters, including the client id and secret, the code, the grant type and the redirect URI. The request is made to the *https://accounts.google.com/o/oauth2/v2/auth* endpoint.

### **POST /authenticate**

It is used to manage interactions between the **RADIUS** server and the auth server. The */authenticate* endpoint verifies if the tokens provided in the received request (ID token and access token) match the stored tokens in RocksDB. The tokens are received in a JSON format that contains the *idToken* and the *accessToken*. The server then searches for the provided *idToken* in the database and, if found, retrieves the associated access token. Finally, it compares the provided *accessToken* with the stored one. The possible responses from the server are "200 OK" if authentication succeeds, "401 Unauthorized" if authentication fails and "415 Unsupported Media Type" if content type is not JSON. The token verification is done in the *authenticateTokens* function. It checks if the tokens correspond to a valid and authenticated user and checks if the access token is still valid and not expired. If the access token is expired, it calls the *exchangeRefreshTokenForNewAccessToken* function to replace it and notify the device that the access token was refreshed via Firebase Messaging [50]. Firebase Cloud Messaging allows the auth server to reliably send messages to a client application. The client application is identified via a unique token string which is required to send the message to the client. When the client receives the message, he can then update its access token with the refreshed one and make a new access request to the network with the valid tokens.

Here is the pseudo-code for the *authenticateTokens* function:

---

**Algorithm 1** authenticateToken: Checks token validity
 

---

**Data:** idToken, accessToken**Result:** true or falseStoredIdToken  $\leftarrow$  GetIdTokenFromDB(idToken)StoredAccessToken  $\leftarrow$  GetAccessTokenFromDB(idToken)**if** *StoredIdToken* is null **then**| **return** false**end****if** *isAccessTokenValid()* is false **then**StoredRefreshToken  $\leftarrow$  GetRefreshTokenFromDB(idToken) **if** *StoredRefreshToken*  $\neq$  null **and** *accessToken* = *hashed(StoredAccessToken)* **then**| newAccessToken  $\leftarrow$  *exchangeRefreshTokenForNewAccessToken(StoredRefreshToken)*

| StoreNewAccessTokenInDB(newAccessToken)

| NotifyDeviceApplication() // FirebaseMessaging

| **return** false| **end**| **return** false**end****if** *hashed(StoredAccessToken)* = *accessToken* **then**| **return** true**end**


---

The *GetIdTokenFromDB*, *StoredAccessToken* and *GetRefreshTokenFromDB* functions simply retrieve the tokens from the embedded database of the auth server.

The *isAccessTokenValid* function is used to check if the access token is not expired by querying the *https://oauth2.googleapis.com/tokeninfo?access\_token=\$accessToken* endpoint from Google and checking the "expires\_in" field from the JSON response.

The *exchangeRefreshTokenForNewAccessToken* function is used to exchange the expired access token for a new and refreshed access token using the refresh token. The request is made to the usual *https://oauth2.googleapis.com/token* endpoint with the grant\_type of the request set to "refresh\_token".

## GET /generateAndroidProfile

It is used to generate Android profile for the users. The /generateAndroidProfile generates a Passpoint (Hotspot 2.0) profile in XML format for Android devices according to the Passpoint specification [6]. It combines the generated profile with a CA certificate of the server into a multipart/mixed response. The profile is filled via the value provided in the AndroidProfileRequest object that contains all required attributes. These attributes are either provided by the user in the GET request or hardcoded. The generateAndroidProfileXML function is called to generate the profile XML by replacing placeholders in the template XML Android profile file. The CA certificate is encoded in Base64 using the loadAndEncodeDERCertificate function. The complete structure, including the headers,

the profile, the CA and the boundary to delimit content is put in a multipart/mixed response and base64-encoded before sending it to the device. The structure of the profile is shown in Figure 5.10.

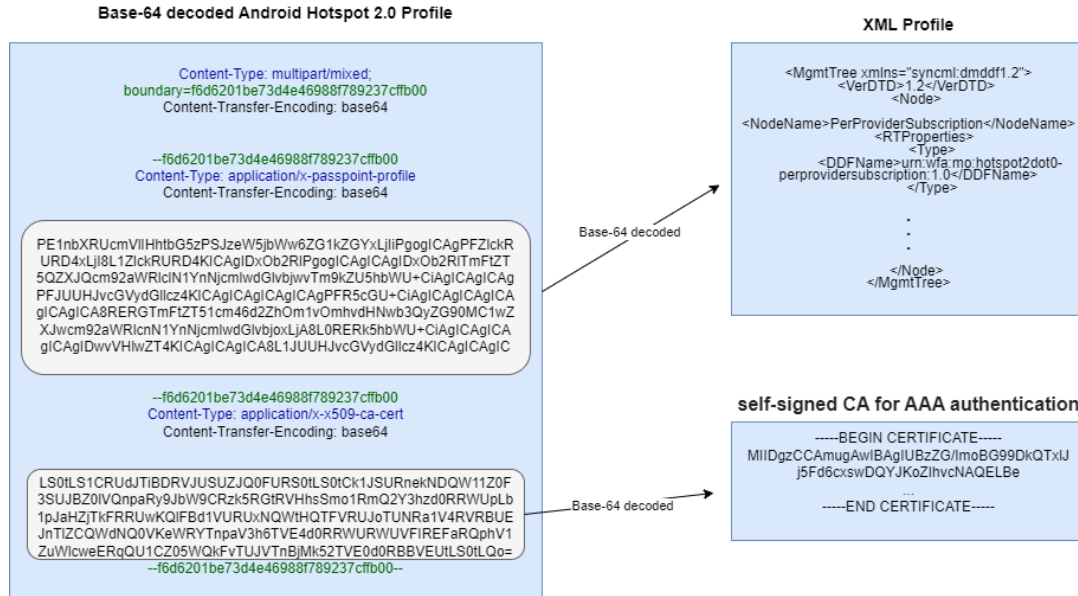


Figure 5.10: The Passpoint profile based on the structure described in [6]

## Other endpoints

For testing and demonstration purposes only, other endpoints are defined, but should not be used in production:

- POST /addUser: It adds a new user (tokens) to the database via a JSON containing idToken, accessToken, and optionally refreshToken.
- PUT /updateUser: It updates stored tokens for an existing user.
- DELETE /deleteUser: It deletes tokens for a user identified by idToken.

### 5.6.2 Token Handling

Tokens are stored in a RocksDB database using the idToken as the key. The accessToken, refreshToken and FMCToken are concatenated with :: as the delimiter. The database supports operations such as storing (put), retrieving (get), and deleting (delete) tokens.

## Persistence

The user data are stored in RocksDB, which is a persistent key-value database. The RocksDB database is initialized on a specific path (data/rocksdb in the code). This directory is created during the first run if it does not exist, and the database is persistent. Even if the server restarts, the data remain in the data/rocksdb directory unless explicitly deleted.

### 5.6.3 External Service: Google OAuth Token Exchange

The *exchangeAuthCodeForTokens* function handles the exchange of an authorization code for tokens using the Google OAuth endpoint. It uses OkHttp with logging for HTTP requests.

The *exchangeRefreshTokenForNewAccessToken* function allows the server to refresh an expired access token using the refresh token.

### 5.6.4 Formats

The Request/Response Format is JSON. In kotlin, it is possible to define structures using data classes annotated with `@Serializable`, to handle authentication requests and responses on the server. The `@Serializable` annotation enables seamless serialization and deserialization of these data classes into JSON and vice versa, which is particularly useful for handling HTTP requests and responses in a REST API. In this project, four classes are defined:

- `AuthRequest`: "authCode": "string", "fcmToken": "string": It represents the request payload sent to the `/auth` endpoint.
- `AuthResponse`: "accessToken": "string", "refreshToken": "string?", "idToken": "string": It represents the response payload returned by the `/auth` endpoint after a successful token exchange.
- `AuthTokens`: "idToken": "string", "accessToken": "string" : It represents the payload for the `/authenticate` and `/deleteUser`
- `AuthTokensWithRefresh`: "idToken": "string", "accessToken": "string", "refreshToken": "string?" : It represents the payload for the `/addUser` and `/updateUser` endpoints.
- `AndroidProfileRequest`: "friendlyName": "string", "fqdn": "string", "realm": "string", "username": "string", "password": "string", "eapType": "int", "innerMethod": "string" : It represents the attributes needed to construct the Android Passpoint profile. )

### 5.6.5 SSL configuration

To set up and start the server with SSL (HTTPS), Ktor is used. SSL is mandatory as endpoints are accessible only via HTTPS [34]. In the main server function, the Ktor server is initialized with an SSL configuration, loading a PKCS12 keystore (server.p12) for HTTPS. The server is then started on port 443.

The SSL configuration involves certificates. These certificates were generated using openssl in the same way as in the subsection 5.5.1. This will result in a server.pem file that is the server certificate. Its SAN is *marie.tiedie.io*.

Kotlin applications often use the PKCS12 format for keystores in Netty SSL configurations with Ktor, as it is the default keystore type in modern JVM environments [97]. To convert the certificates to PKCS12 format, this openssl command is used:

```
openssl pkcs12 -export -in server.pem -inkey server.key -out server.p12
-name ktorServer -CAfile ca.pem -caname root
```

This command assigns an alias (ktorServer) to the key pair in the keystore and generates the server.p12 certificate.

## 5.7 IDP (Firebase, Google)

### 5.7.1 Firebase set up

Firebase requires several setups before actually being able to authenticate users through Google on Android. This is documented on the "Authenticate with Google on Android" page of the Firebase documentation [51].

First, a Firebase project must be created via the Firebase console [32]. In the authentication section of the new project, a sign-in method must be added and in this case, Google is selected. With Google authentication, the SHA fingerprint of the application must be specified. The fingerprint can be found via the Gradle signingReport command in Android Studio, as can be seen in Figure 5.11.

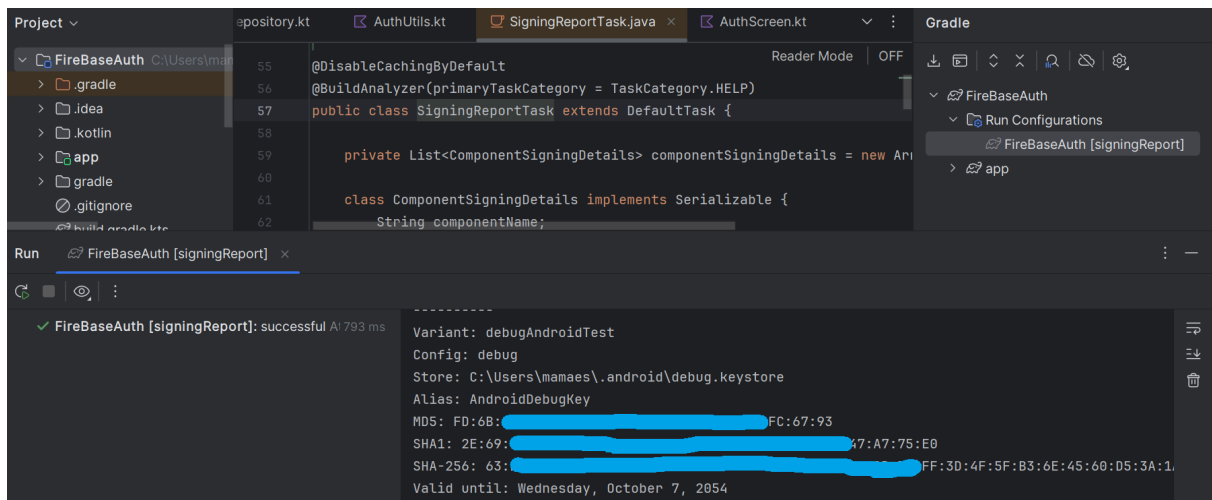


Figure 5.11: The SHA-1 of the signing certificate using the Gradle signingReport command in Android Studio

After this step, the Firebase configuration file (google-services.json) needs to be replaced in the Android Studio project with the new one provided in Firebase.

Some dependencies are needed in the root-level (project-level) Gradle file to configure Google services plugin such as Firebase and Google APIs in the application:

```
alias(libs.plugins.google.gms.google.services) apply false
```

Other dependencies are needed in the module (app-level) Gradle file to apply the Google services plugin and include dependencies for Firebase Authentication, Google Play Services Authentication, and OkHttp (for HTTP requests):

```
alias(libs.plugins.google.gms.google.services)

implementation(libs.firebase.auth)
implementation(platform(libs.firebase.bom))
implementation(libs.google.firebase.auth)
implementation(libs.play.services.auth)
implementation(libs.okhttp)
```

Note that using Firebase BoM ensures that all Firebase dependencies are compatible and have consistent versions throughout the project.

After this, everything is setup and Firebase can be integrated in the mobile application.

In this project, Android Studio [33] will be used to develop the Kotlin application. It is particularly useful as a Firebase Assistant tool in it allows to directly add the firebase project to the Android project. It also allows to emulate an Android phone.

### 5.7.2 Google Identity set up

To use the Google OAuth 2.0 endpoints and thus get the necessary tokens, Google Identity must be correctly set up, as documented in the "Using OAuth 2.0 for Web Server Applications" page of the documentation [34]. The Google APIs must be enabled in the API Console [5] for this project. For this project, the Identity Toolkit API [25] to manage authentication through the Identity Platform and Firebase Installations API [51] that help with application development are enabled.

In the console, Firebase autocreated the project, including the API keys, OAuth 2.0 client ID and service accounts. In the OAuth client ID page section, the Web client (auto created by Google Service) needs to be modified to allow a new redirect URI, *https://marie.tiedie.io/auth*. In this section, the client id and secret used for the authorization and token requests from the auth server are available, as can be seen in Figure 5.12.

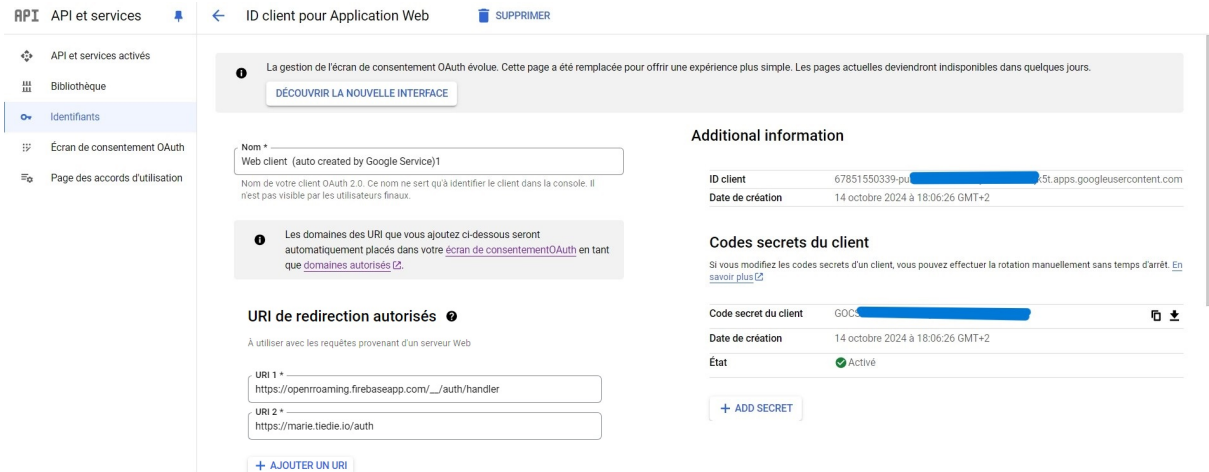


Figure 5.12: Web client OAuth 2.0 ID from API Console, taken from [5]

The client ID and secret will be used on the auth server to make requests to the IDP.

## 5.8 IDP (BOSA, e-ID)

The integration with BOSA and e-ID credentials is not yet done because it requires a client ID and a client secret, which must be provided by BOSA. Due to legal and administrative procedures, the process to receive these credentials has taken longer than expected and cannot be documented here.

However, the overall integration logic is essentially the same as with Firebase, Google Identity and Google credentials. Firebase is designed to support many types of credentials. As a result, since the integration is already working with Google credentials, the transition to using e-ID credentials will involve minimal changes. The biggest modification required will be to update the client ID and client secret to the ones provided by BOSA, and update the endpoint to query.

## 5.9 Demonstration

A video demonstrating the whole solution is available at the following link:

OpenRoaming thesis - Demo and progress (Marie Maes)-20241219 1405-1.mp4

First, the client must authenticate through the IDP using either its Google or e-ID credentials in the mobile application, as shown in Figure 5.15.

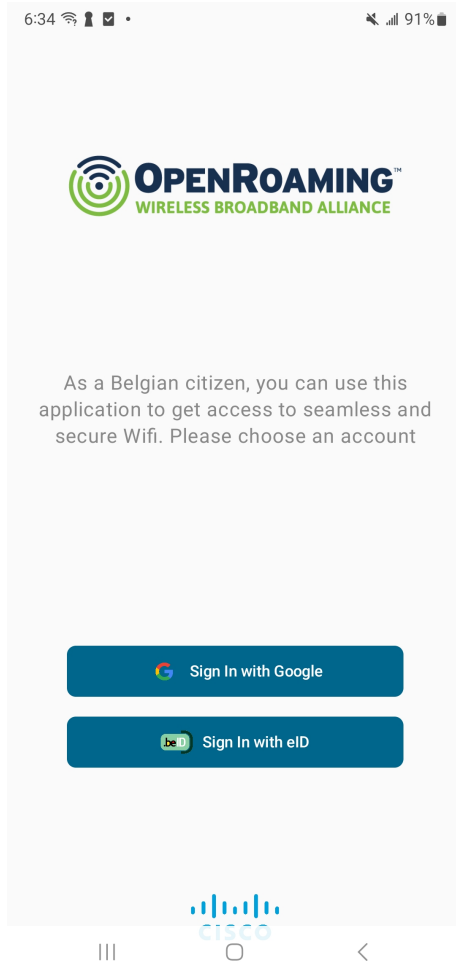


Figure 5.13: Welcome page of the mobile application that allows the user to authenticate

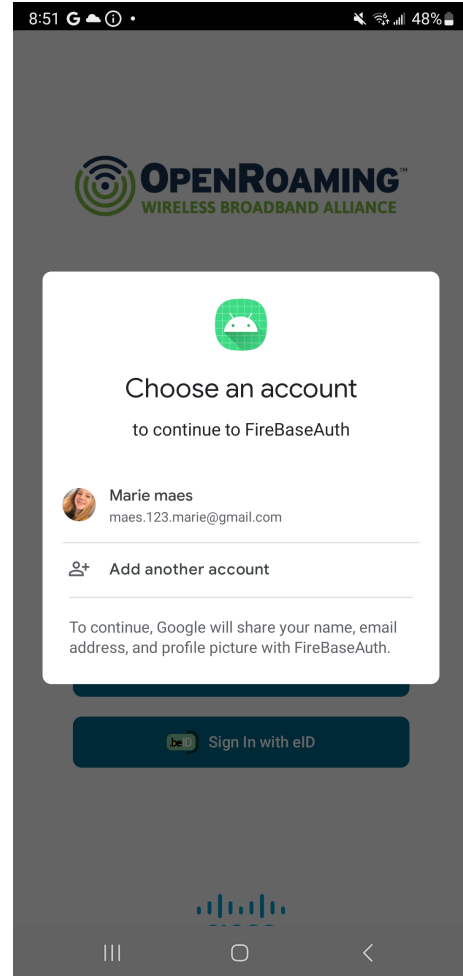


Figure 5.14: Google screen to authenticate via the FirebaseAuth application

Figure 5.15: Screenshots of the mobile application

Once the user is successfully authenticated, he will receive an authorization code from either the Google or e-ID IDP, and will exchange it for an id, access and refresh tokens by making a request containing the authorization code to the auth endpoint of the auth server. The auth server will then manage the exchange of the authorization code for the tokens via the Google or e-ID endpoint, as can be seen in Figure 5.16. The mobile application then receives the tokens and sends another request to the auth server to get its Hotspot 2.0 profile through the generateAndroidProfile endpoint.



## CHAPTER 5. PROTOTYPE: IMPLEMENTATION AND DEMONSTRATION

[illegible]

Figure 5.16: Logs of the auth server that show the requests to the auth and to the generateAndroidProfile endpoint

Once the mobile application receives a response from the auth server with the profile, the user will see a pop-up notification in the settings asking if he wants to allow suggested Wi-Fi networks, which will authorize the mobile application to suggest networks and thus connect the device automatically and seamlessly, as can be seen in Figure 5.17.

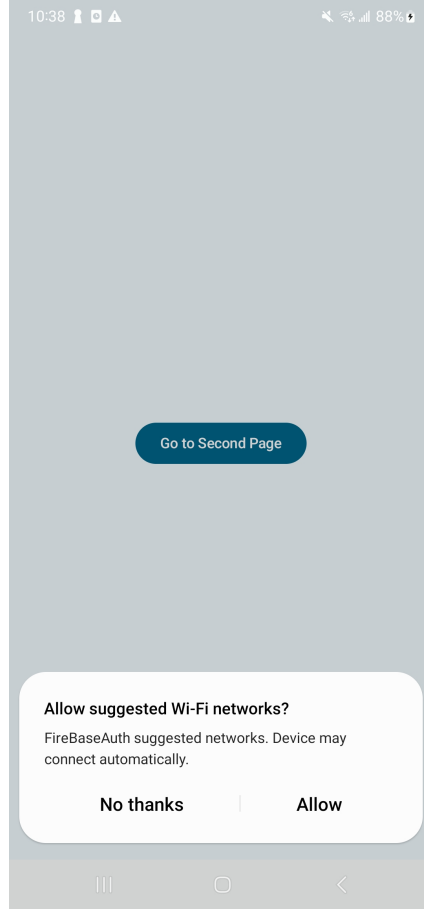


Figure 5.17: Pop-up notification to allow suggested Wi-Fi networks

Once this is done, the device can now seamlessly connect to the **OpenRoaming**-enabled Wi-Fi. It will involve all the components developed in this project. These components are represented in Figure 5.18.

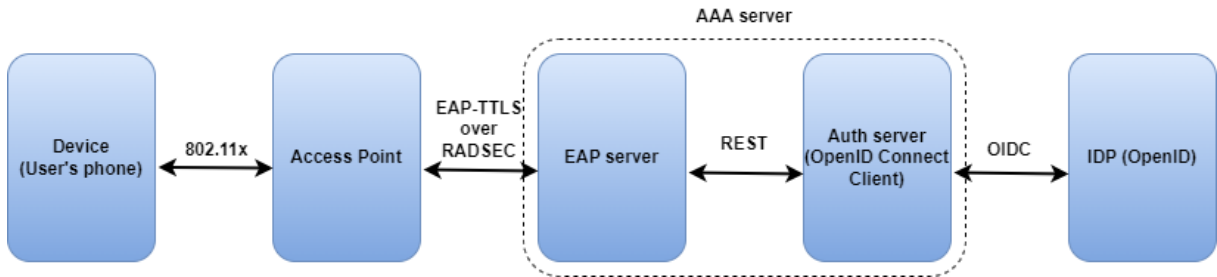


Figure 5.18: OpenRoaming components summary

First, the client device will broadcast probe requests to search for available SSIDs near him. The access points then respond with probe responses. One of these responses is sent from the AP with SSID "Test1" and is shown in Figure 5.19.

The "Wi-Fi Alliance: Hotspot 2.0 Indication" field indicates that the network supports Passpoint (802.11u/Hotspot 2.0). Then, the "Microsoft Corp." and "Cisco Meraki" Vendor Specific fields indicate the **OpenRoaming** RCOI for "allow all users", 004096. The interworking element is part of the 802.11u specification and supports network discovery

and selection by providing additional metadata about the network. The Access Network Type being "Free public network" is common for OpenRoaming-enabled networks. Finally, the advertisement protocol is set to the Access Network Query Protocol (ANQP), which is the common protocol used with OpenRoaming to query the network for information such as roaming consortiums and other OpenRoaming-related fields.

```

> Frame 16311: 425 bytes on wire (3400 bits), 425 bytes captured (3400 bits)
> Radiotap Header v0, Length 48
> 802.11 radio information
> IEEE 802.11 Probe Response, Flags: .....
- IEEE 802.11 Wireless Management
  - Fixed parameters (12 bytes)
  - Tagged parameters (341 bytes)
    - Tag: SSID parameter set: "Test1"
    - Tag: Vendor Specific: Wi-Fi Alliance: P2P
    - Tag: Vendor Specific: Wi-Fi Alliance: Hotspot 2.0 Indication
    - Tag: Vendor Specific: Microsoft Corp.: WMM/WME: Parameter Element
    - Tag: Vendor Specific: Cisco Meraki
    - Tag: Vendor Specific: Cisco Systems Inc: Aironet CCX version = 5
      Tag Number: Vendor specific (221)
      Tag length: 5
      OUI: 00:40:96 (Cisco Systems, Inc)
      Vendor Specific OUI type: 3
      Aironet IE type: CCX version (3)
      Aironet IE CCX version: 5
    - Tag: Interworking
      Tag Number: Interworking (107)
      Tag length: 3
      ... 0011 = Access Network Type: Free public network (3)
      ... 1 .... = Internet: 1
      ... 0 .... = ASRA: 0
      ... 0 .... = ESR: 0
      ... 0 .... = UESA: 0
      Venue Group: Unspecified (0)
      Venue Type: 0
    - Tag: Advertisement Protocol
      Tag Number: Advertisement Protocol (108)
      Tag length: 2
    - Advertisement Protocol element: ANQP
      - Advertisement Protocol Tuple: Access Network Query Protocol
        .111 1111 = Query Response Length Limit: 127
        0... .... = PAME-BI: 0
        Advertisement Protocol ID: Access Network Query Protocol (0)
    
```

Figure 5.19: Probe response information for the OpenRoaming-enabled SSID Test1

The Meraki access point then receives the 802.11 association request from the client, as shown in Figure 5.20.

Dec 20 07:02:07	Marie AP	Test1	Marie-s-Note20-Ultra	802.1X	802.1X authentication	radio: 1, vap: 2, client_mac: 3E:A8:EA:5B:B9:6C <a href="#">more »</a>
Dec 20 07:02:07	Marie AP	Test1	Marie-s-Note20-Ultra	802.1X	Successful authentication (EAP success)	radio: 1, vap: 2, client_mac: 3E:A8:EA:5B:B9:6C <a href="#">more »</a>
Dec 20 07:02:07	Marie AP	Test1	Marie-s-Note20-Ultra	802.1X	RADIUS response	radio: 1, vap: 2, group: <a href="#">more »</a>
Dec 20 07:02:03	Marie AP	Test1	Marie-s-Note20-Ultra	802.1X	RADIUS response	radio: 1, vap: 2, group: <a href="#">more »</a>
Dec 20 07:02:03	Marie AP	Test1	Marie-s-Note20-Ultra	802.11	802.11 association	channel: 104, rssi: 23, band: 5

Figure 5.20: Event logs from the meraki access point

When the access point receives the request from a user to gain access to the network, it needs to communicate with the RADIUS server for the authentication process via EAP-TTLS. First, the EAP identity is sent to the RADIUS server, as can be seen in Figure 5.24. The RADIUS server receives an Access-Request packet with the anonymous outer identity set to:

User-Name = "anonymous@test-beid.openroaming.net"

The User-Name is "anonymous" with the realm being *test – beid.openroaming.net*. The NAS-IP-Address is the IP address of the access point. There are also some attributes specific to Meraki, such as the device name of the access point. In this case, the Meraki access point is "Marie AP".

```
(0) Received Access-Request Id 32 from 52.48.33.123:59900 to 185.48.12.253:2083 length 442
(0) User-Name = "anonymous@test-beid.openroaming.net"
(0) NAS-IP-Address = 192.168.1.54
(0) NAS-Identifier = "E4-55-A8-1F-DE-D4:vap2"
(0) Called-Station-Id = "EE-55-B8-1F-DE-D4:Test1"
(0) NAS-Port-Type = Wireless-802.11
(0) Service-Type = Framed-User
(0) NAS-Port = 1
(0) Calling-Station-Id = "3E-A8-EA-5B-B9-6C"
(0) Connect-Info = "CONNECT 54.00 Mbps / 802.11ax / RSSI: 20 / Channel: 104"
(0) Acct-Session-Id = "3F0EE5A73824FB03"
(0) Acct-Multi-Session-Id = "46C939B080680AFA"
(0) WLAN-Pairwise-Cipher = 1027076
(0) WLAN-Group-Cipher = 1027076
(0) WLAN-AKM-Suite = 1027073
(0) WLAN-Group-Mgmt-Cipher = 1027078
(0) Meraki-Network-Name = "Marie network - wireless"
(0) Meraki-AP-Name = "Marie AP"
(0) Operator-Name = "417988.dnaspaces.cisco:US"
(0) Meraki-Device-Name = "Marie AP"
(0) Framed-MTU = 1400
(0) EAP-Message = 0x0230002801616e6f6e796d6f757340746573742d626569642e6f70656e726f616d696e6726e6e6574
(0) HS20-AP-Version = 2
(0) Message-Authenticator = 0x5d0a8bb829669fa0b49408cb41889c1f
(0) Chargeable-User-Identity = 0x
```

Figure 5.21: EAP Identity exchange in the Access-Request RADIUS message

After the initial Access-Request packet, the RADIUS server sends an Access-Challenge to actually start the TTLS conversation, as shown in Figure 5.22.

```
(0) Sent Access-Challenge Id 32 from 185.48.12.253:2083 to 52.48.33.123:59900 length 64
(0) EAP-Message = 0x013100061520
(0) Message-Authenticator = 0x00000000000000000000000000000000
(0) State = 0x5431ec645400f9fae984eef0ef513b1a
(0) Finished request
```

Figure 5.22: Access-Request packet from the RADIUS server

This TLS handshake between the AP and the RADIUS server then starts and can be captured using tcpdump and filtering on the tcp port 2083 (RADSEC default port). It is represented in Figure 5.23.

No.	Time	Source	Destination	Protocol	Length	802.1Q Virtual LAN	ID	Info
280	16.808318	81.245.174.125	185.48.12.253	TCP	74			36021 → 2083 [SYN] Seq=0 Win=31944 Len=0 MSS=1452 SACK_PERM TSval=2428261675 TSecr=2428261675
281	16.808379	185.48.12.253	81.245.174.125	TCP	74			2083 → 36021 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=7 TSecr=2428261675
282	16.824024	81.245.174.125	185.48.12.253	TCP	66			36021 → 2083 [ACK] Seq=1 Ack=1 Win=32000 Len=0 TSval=2428261691 TSecr=72300777
284	16.827424	81.245.174.125	185.48.12.253	TLSv1.2	244			Client Hello
285	16.827450	185.48.12.253	81.245.174.125	TCP	66			2083 → 36021 [ACK] Seq=1 Ack=179 Win=65024 Len=0 TSval=723007796 TSecr=2428261
289	16.830867	185.48.12.253	81.245.174.125	TLSv1.2	1308			Server Hello, Certificate, Server Key Exchange, Server Hello Done
291	16.846224	81.245.174.125	185.48.12.253	TCP	66			36021 → 2083 [ACK] Seq=179 Ack=1243 Win=31360 Len=0 TSval=2428261713 TSecr=723
292	16.848777	81.245.174.125	185.48.12.253	TLSv1.2	192			Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
293	16.849670	185.48.12.253	81.245.174.125	TLSv1.2	117			Change Cipher Spec, Encrypted Handshake Message
294	16.864842	81.245.174.125	185.48.12.253	TCP	66			36021 → 2083 [ACK] Seq=305 Ack=1294 Win=31360 Len=0 TSval=2428261732 TSecr=723
295	16.871263	81.245.174.125	185.48.12.253	TLSv1.2	424			Application Data
296	16.872565	185.48.12.253	81.245.174.125	TLSv1.2	164			Application Data
297	16.887740	81.245.174.125	185.48.12.253	TCP	66			36021 → 2083 [ACK] Seq=663 Ack=1392 Win=31360 Len=0 TSval=2428261755 TSecr=723
298	16.907557	81.245.174.125	185.48.12.253	TLSv1.2	595			Application Data
299	16.911278	185.48.12.253	81.245.174.125	TLSv1.2	1164			Application Data
300	16.933224	81.245.174.125	185.48.12.253	TLSv1.2	440			Application Data
301	16.934225	185.48.12.253	81.245.174.125	TLSv1.2	422			Application Data
302	16.967839	81.245.174.125	185.48.12.253	TLSv1.2	570			Application Data
303	16.969609	185.48.12.253	81.245.174.125	TLSv1.2	219			Application Data
304	17.021515	81.245.174.125	185.48.12.253	TLSv1.2	509			Application Data
305	17.062400	185.48.12.253	81.245.174.125	TCP	66			2083 → 36021 [ACK] Seq=2999 Ack=2513 Win=64128 Len=0 TSval=723008031 TSecr=242
306	17.749423	185.48.12.253	81.245.174.125	TLSv1.2	265			Application Data
307	17.809220	81.245.174.125	185.48.12.253	TCP	66			36021 → 2083 [ACK] Seq=2513 Ack=3198 Win=31360 Len=0 TSval=2428262677 TSecr=72

Figure 5.23: The TLS Tunnel establishment between the Meraki AP and the RADIUS server, in Wireshark

The same exchange is also captured in the FreeRADIUS logs, as can be seen in Figure 5.24.

```

(4) eap: Peer sent packet with method EAP TTLS (21)
(4) eap: Calling submodule eap_ttls to process data
(4) eap_ttls: Authenticate
(4) eap_ttls: (TLS) EAP Done initial handshake
(4) eap_ttls: (TLS) TTLS - Handshake state - Server SSLv3/TLS write server done
(4) eap_ttls: (TLS) TTLS - Handshake state - Server SSLv3/TLS read client key exchange
(4) eap_ttls: (TLS) TTLS - Handshake state - Server SSLv3/TLS read change cipher spec
(4) eap_ttls: (TLS) TTLS - Handshake state - Server SSLv3/TLS read finished
(4) eap_ttls: (TLS) TTLS - send TLS 1.2 ChangeCipherSpec
(4) eap_ttls: (TLS) TTLS - Handshake state - Server SSLv3/TLS write change cipher spec
(4) eap_ttls: (TLS) TTLS - send TLS 1.2 Handshake, Finished
(4) eap_ttls: (TLS) TTLS - Handshake state - Server SSLv3/TLS write finished
(4) eap_ttls: (TLS) TTLS - Handshake state - SSL negotiation finished successfully
(4) eap_ttls: (TLS) TTLS - Connection Established
(4) eap_ttls: TLS-Session-Cipher-Suite = "ECDHE-RSA-AES128-GCM-SHA256"
(4) eap_ttls: TLS-Session-Version = "TLS 1.2"
(4) eap: Sending EAP Request (code 1) ID 52 length 61
(4) eap: EAP session adding &reply:State = 0x5431ec645705f9fa
(4) [eap] = handled
(4) } # Auth-Type EAP = handled

```

Figure 5.24: The TLS Tunnel establishment between the Meraki AP and the RADIUS server, in the FreeRADIUS logs

This handshake concludes the first phase of the EAP-TTLS conversation. After that, the tunnel is successfully established and the second phase starts. In this phase, the client needs to authenticate by giving its credentials as a username and password to the RADIUS server. This exchange is shown in figure Figure 5.25.

```

(5) eap: Peer sent packet with method EAP TTLS (21)
(5) eap: Calling submodule eap_ttls to process data
(5) eap_ttls: Authenticate
(5) eap_ttls: (TLS) EAP Done initial handshake
(5) eap_ttls: Session established. Proceeding to decode tunneled attributes
(5) eap_ttls: Got tunneled request
(5) eap_ttls: User-Name = "108441824109544107469"
(5) eap_ttls: User-Password = "BNW43 3j7 cH0vRfsebFJEqR00Jw0CHk8fKYPXjnQdM="
(5) eap_ttls: FreeRADIUS-Proxyed-To = 127.0.0.1
(5) eap_ttls: Sending tunneled request
(5) Virtual server inner-tunnel received request
(5) User-Name = "108441824109544107469"
(5) User-Password = "BNW43 3j7 cH0vRfsebFJEqR00Jw0CHk8fKYPXjnQdM="
(5) FreeRADIUS-Proxyed-To = 127.0.0.1
(5) NAS-IP-Address = 192.168.1.54
(5) NAS-Identifier = "E4-55-A8-1F-DE-D4:vap2"
(5) Called-Station-Id = "EE-55-B8-1F-DE-D4:Test1"
(5) NAS-Port-Type = Wireless-802.11
(5) Service-Type = Framed-User
(5) NAS-Port = 1
(5) Calling-Station-Id = "3E-A8-EA-5B-B9-6C"
(5) Connect-Info = "CONNECT 54.00 Mbps / 802.11ax / RSSI: 23 / Channel: 104"
(5) Acct-Session-Id = "3F0EE5A73824FB03"
(5) Acct-Multi-Session-Id = "46C939B080680AFA"
(5) WLAN-Pairwise-Cipher = 1027076
(5) WLAN-Group-Cipher = 1027076
(5) WLAN-AKM-Suite = 1027073
(5) WLAN-Group-Mgmt-Cipher = 1027078
(5) Meraki-Network-Name = "Marie network - wireless"
(5) Meraki-AP-Name = "Marie AP"
(5) Operator-Name = "417988.dnaspaces.cisco:US"
(5) Meraki-Device-Name = "Marie AP"
(5) Framed-MTU = 1400
(5) HS20-AP-Version = 2
(5) Chargeable-User-Identity = 0x

```

Figure 5.25: Second phase of the EAP-TTLS exchange: the user sends its credentials

## CHAPTER 5. PROTOTYPE: IMPLEMENTATION AND DEMONSTRATION

The RADIUS server then needs to ensure that the credentials given by the user are valid. To do so, it sends a REST request to the *authenticate* endpoint of the auth server with the received id token and access token, as shown in Figure 5.26.

```
(5) Found Auth-Type = REST
(5) # Executing group from file /etc/freeradius/3.0/sites-enabled/inner-tunnel
(5) Auth-Type REST {
rlm_rest(rest): Reserved connection (0)
(5) rest: Expanding URI components
(5) rest: EXPAND https://marie.tiedie.io:443
(5) rest: --> https://marie.tiedie.io:443
(5) rest: EXPAND /authenticate
(5) rest: --> /authenticate
(5) rest: Sending HTTP POST to "https://marie.tiedie.io:443/authenticate"
(5) rest: EXPAND { "idToken": "%{User-Name}", "accessToken": "%{User-Password}" }
(5) rest: --> { "idToken": "108441824109544107469", "accessToken": "BNW43 3j7 cH0vRfsebFJEqR00Jw0CHk8fKYPXjnQdM=" }
```

Figure 5.26: REST call by the RADIUS server to the auth server to check the user credentials

Upon receiving the request, the auth server checks if the credentials received (the id and access token) are in its database. If this is the case and the tokens are valid, it responds to the RADIUS server with a "200 OK" status, as shown in Figure 5.27.

```
2024-12-20 06:02:07.697 [eventLoopGroupProxy-4-8] INFO Application - Received /authenticate request: AuthTokens(idToken
=108441824109544107469, accessToken=BNW43 3j7 cH0vRfsebFJEqR00Jw0CHk8fKYPXjnQdM=)
2024-12-20 06:02:07.801 [eventLoopGroupProxy-4-8] INFO Application - EXPIRE IN INFO : 3578
2024-12-20 06:02:07.801 [eventLoopGroupProxy-4-8] INFO Application - Authentication successful for ID Token 1084418241
09544107469
2024-12-20 06:02:07.804 [eventLoopGroupProxy-4-8] INFO Application - 200 OK: POST - /authenticate
```

Figure 5.27: Second phase of the EAP-TTLS exchange: the user sends its credentials

The "200 OK" status is received by the RADIUS server that interprets it as a successful authentication for the user, and thus sends an Access-Accept message to the access point.

```
(5) Virtual server sending reply
(5) REST-HTTP-Status-Code := 200
(5) eap_ttls: Got tunneled Access-Accept
(5) eap: Sending EAP Success (code 3) ID 52 length 4
(5) eap: Freeing handler
(5) [eap] = ok
(5) } # Auth-Type EAP = ok
(5) # Executing section post-auth from file /etc/freeradius/3.0/sites-enabled/inner-tunnel
(5) post-auth {
(5) if (0) {
(5) if (0) -> FALSE
(5) } # post-auth = noop
(5) Sent Access-Accept Id 34 from 185.48.12.253:2083 to 52.48.33.123:28832 length 197
(5) MS-MPPE-Recv-Key = 0xfc4199479f2dfc46518cbfb26fd08549069465241b7b99aeafa5186f9ff9032
(5) MS-MPPE-Send-Key = 0x9745146c000b595cfbcad56227b08d487492a94f401445d2f2dc9c18bb1dbe09
(5) EAP-Message = 0x03340004
(5) Message-Authenticator = 0x00000000000000000000000000000000
(5) User-Name = "anonymous@test-beid.openroaming.net"
(5) Finished request
```

Figure 5.28: Access-Accept packet from the RADIUS server

When the access point receives the Access-Accept message, its next logs (RADIUS response, EAP success) confirm that the access point communicated with the RADIUS server for authentication using EAP successfully. Finally, the 802.1X log shows that the client, when successfully authenticated with the RADIUS server, authenticates over the WPA2/WPA3 protocol, as shown in Figure 5.29.



Dec 20 07:02:07	Marie AP	Test1	Marie-s-Note20-Ultra	802.1X	802.1X authentication	radio: 1, vap: 2, client_mac: 3E:A8:EA:5B:B9:6C <a href="#">more »</a>
Dec 20 07:02:07	Marie AP	Test1	Marie-s-Note20-Ultra	802.1X	Successful authentication (EAP success)	radio: 1, vap: 2, client_mac: 3E:A8:EA:5B:B9:6C <a href="#">more »</a>
Dec 20 07:02:07	Marie AP	Test1	Marie-s-Note20-Ultra	802.1X	RADIUS response	radio: 1, vap: 2, group: <a href="#">more »</a>
Dec 20 07:02:03	Marie AP	Test1	Marie-s-Note20-Ultra	802.1X	RADIUS response	radio: 1, vap: 2, group: <a href="#">more »</a>
Dec 20 07:02:03	Marie AP	Test1	Marie-s-Note20-Ultra	802.11	802.11 association	channel: 104, rssi: 23, band: 5

Figure 5.29: Event logs from the meraki access point

Finally, when the access point gives access to the user to the network, the user will finally connect to the Wi-Fi without requiring any action on his side. The user did not push any button, enter any credentials, had to accept anything, etc. Once authenticated with Google or e-ID using the application, he never has to authenticate again and will connect seamlessly to the available OpenRoaming Wi-Fi. In fact, Figure 5.30 shows that the user is "Connected via FirebaseAuth", which is the mobile application.

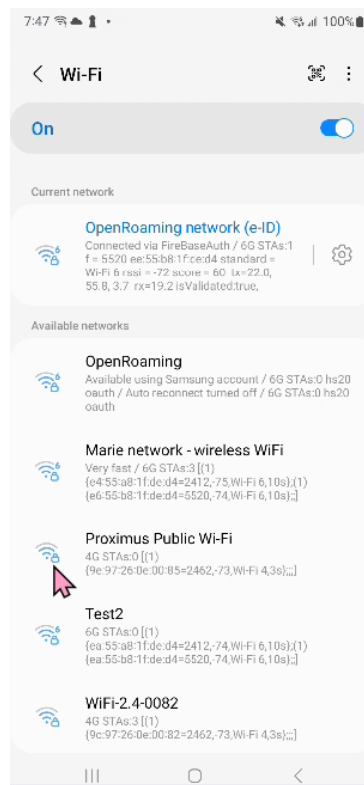


Figure 5.30: Screenshot of the user being connected to an OpenRoaming Wi-Fi

## 5.10 Use-cases

In municipality or communal building, such as a town hall, public library, or community center, free Wi-Fi access is often offered to their visitors. These venues often have a wide range of users, including residents, tourists, and government workers, making it essential to provide secure and seamless connectivity.

By integrating OpenRoaming with e-ID credentials, these buildings enable visitors to authenticate with a highly secure identity verification mechanism, ensuring that only legitimate users gain access to the Wi-Fi network and making it difficult for unauthorized users to access the network. Also note that authenticating users with e-ID credentials

## CHAPTER 5. PROTOTYPE: IMPLEMENTATION AND DEMONSTRATION

ensures accountability, as each user is tied to a verified identity. It could thus create a safer and more reliable **Wi-Fi** environment for every visitor.



# Chapter 6

## Conclusion

The main objective of this project was to evaluate how **e-ID** can be integrated as an Identity Provider (IDP) within the **OpenRoaming** federation, in order to get seamless, secure, and private **Wi-Fi** access. The project has focused on understanding the theoretical background of **OpenRoaming**, **Wi-Fi** standards, and network components, as well as exploring the **e-ID** authentication method.

Once the theoretical background was established, it was necessary to investigate how this integration could possibly be implemented. All components necessary for this project were developed, such as the mobile application, the access point, the **AAA** server which includes the **EAP/RADIUS** server and the auth server, and the **IDP**. Some choices were critical during this step, such as the selection of the **EAP** method and the programming language. The implementation of each component has been carefully thought out, going from the parameters for the Hotspot 2.0 profile to the query parameters for the **IDP** endpoints.

Finally, the actual implementation was done. It combined the configuration for the access point, the **EAP** server, and programming for the mobile application and auth server. It also involved working with certificates and CAs for the **TLS** conversation between the **AP**, the **EAP** server, the auth server and the mobile device.

The final prototype is an Android device that, after downloading the mobile application and authenticating successfully with the **IDP**, is able to seamlessly connect to a **Wi-Fi** that it had never visited before, while remaining in a safe environment.

This secure environment is ensured by several components in the project. First, the configured **FreeRADIUS** server that is able to communicate with the auth server using **HTTPS** for secure user authentication, and that is also able to work with **EAP-TTLS** and **PAP** over **RADSEC** for secure credential transmission with the access point. The access point uses **WPA2 Enterprise** to ensure secure wireless communication. Finally, the use of the **OAuth 2.0** and **OpenID Connect** protocols ensured a secure authentication through the **IDP**, with reliable **e-ID** credentials. The access and refresh tokens, with the access token expiring after some time, also allowed the process to be more secure.

## 6.1 Possible improvements

While this thesis is a strong foundation for integrating **e-ID** as an Identity Provider within the OpenRoaming federation, it also opens the door to potential enhancements and future work. In the following section, some possible improvements are described.

### Actual **e-ID** integration

A potential improvement for the project would be to complete the integration of **e-ID** credentials once the client ID and the secret are provided by BOSA. This would enable seamless authentication using the Belgian **e-ID** system.

### Database security

A potential improvement for this project would be to enhance the security of the embedded RocksDB database, which currently stores credentials as key-value pairs in plain-text. This setup is simple and working but additional measures could be implemented to ensure better protection of sensitive data, such as encryption, access controls, or integration of more advanced security features. These enhancements would help in the case where the database would be compromised. The goal of this enhancement would be that, in the worst-case scenario of a data breach, the exposed data would be unusable by the attacker.

### EAP-PPT

For this project, EAP-TTLS was used not only for its strong security but also because it was compatible with FreeRADIUS, which is not the case of EAP-PPT. However, EAP-PPT is an EAP type that is particularly suited for use cases such as the OpenRoaming one, as it allows users who will connect to a public network to do so anonymously and securely. EAP-PPT would thus minimize the exposure of user credentials and identities during the authentication process. Implementing this would be a significant challenge, as FreeRADIUS does not support it and significant changes would be required.

### Cross-border **e-ID** integration

Currently, the use of **e-ID** credentials in this project is limited to Belgian citizens. However, future work could be to enable cross-border **e-ID** integration, as many European countries also have **e-ID** systems [4]. This improvement would be an interesting feature, as it would align with the OpenRoaming principle of roaming more efficiently. However, this enhancement would be more focused on the legal aspect, since integrating cross-border **e-ID** would only require using different client id, secrets and querying from different endpoints. The major part of the work would thus be the legal communication.

### iOS mobile application

For now, the project is designed to work exclusively with Android devices. A great improvement would be to develop an iOS application that would allow iOS users to also have access to OpenRoaming using their e-ID credentials.

### Scaling and performances

A big improvement for this project would be to evaluate various components of the system under realistic loads to test the scalability and the performances of the OpenRoaming application.

It would involve establishing the metrics that are interesting to measure and where to measure them. For example, it could be the latency for authentication requests to be processed end-to-end, the number of requests the auth server, the RADIUS server, or the overall system can handle per second, the resource utilization of the *marie.tiedie.io* VM that hosts both servers, the failure rate, to evaluate the maximum number of simultaneous users the system can support, etc.

It would also require testing different scenarios that emulate real-world usage. For example, testing authentication requests and test different scenarios where there is a component failure (e.g., Google IDP unavailability). It could also involve the introduction latency, jitter, and packet loss to mimic real-world connectivity issues.

Finally, it can be interesting to test scalability by finding the maximum load that the system can handle and potentially to identify bottlenecks in the components.

# Bibliography

- [1] 1980. User Datagram Protocol. RFC 768. <https://doi.org/10.17487/RFC0768>
- [2] 1981. Transmission Control Protocol. RFC 793. <https://doi.org/10.17487/RFC0793>
- [3] 2024. Eduroam. <https://eduroam.org/> Accessed: 2024-10-31.
- [4] 2024. *eIDAS Dashboard*. <https://eidas.ec.europa.eu/efda/browse/notification/eid-chapter-contacts> Version: 2.15.1 - 30/10/2024. Managed by the Directorate-General for Digital Services, European Commission..
- [5] 2024. *Google API Console: OpenRoaming Project*. <https://console.cloud.google.com/apis/credentials?inv=1&inv=Abh0nA&project=openrroaming> Accessed 13/12/2024.
- [6] Android. 2024. Passpoint (Hotspot 2.0). [https://source.android.com/docs/core/connect/wifi-passpoint#passpoint\\_r1\\_provisioning](https://source.android.com/docs/core/connect/wifi-passpoint#passpoint_r1_provisioning). Last updated 2024-12-18 UTC., Accessed: 11/09/2024.
- [7] Auth0. 2024. *Introduction to JSON Web Tokens*. <https://jwt.io/introduction> Crafted by Auth0 - JWT.io Token Based Authentication.
- [8] Brinckman B. and Sroga J. 2024. OpenRoaming under the hood. PowerPoint presentation at Cisco Live. Distinguished Engineer, bbrinckm@cisco.com and Network Support Engineer, jsroga@cisco.com.
- [9] Ed. B. de Medeiros, M. Scurtescu, P. Tarjan, and M. Jones. 2014. OAuth 2.0 Multiple Response Type Encoding Practices. [https://openid.net/specs/oauth-v2-multiple-response-types-1\\_0.html](https://openid.net/specs/oauth-v2-multiple-response-types-1_0.html) Google, Facebook, Microsoft, February 25, 2014.
- [10] Mark A. Beadles, Jari Arkko, Dr. Bernard D. Aboba, and Pasi Eronen. 2005. The Network Access Identifier. RFC 4282. <https://doi.org/10.17487/RFC4282>
- [11] Belgian Mobile ID SA/NV. 2024. *itsme - Your Digital ID*. <https://www.itsme-id.com/en-BE> © 2024 Belgian Mobile ID SA/NV .

- [12] Pat R. Calhoun and Dr. Bernard D. Aboba. 2003. RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP). RFC 3579. <https://doi.org/10.17487/RFC3579>
- [13] Pat R. Calhoun, Erik Guttman, Jari Arkko, and John A. Loughney. 2003. Diameter Base Protocol. RFC 3588. <https://doi.org/10.17487/RFC3588>
- [14] Certipost. 2024. *Belgian Certificate Policy Practice Statement for eID PKI Infrastructure Citizen CA*. Technical Report. [https://repository.eid.belgium.be/downloads/citizen/en/CPS\\_CitizenCA\\_BRCA34.pdf](https://repository.eid.belgium.be/downloads/citizen/en/CPS_CitizenCA_BRCA34.pdf) Release Date: 03/09/2024, OIDs: 2.16.56.1.1.1.2, 2.16.56.9.1.1.2, 2.16.56.10.1.1.2, 2.16.56.12.1.1, Initial version 1.0 by Bart Eeman.
- [15] Cisco. 2024. Cisco Spaces. <https://spaces.cisco.com/> © 2024 Cisco. All Rights Reserved.
- [16] Cisco 2024. *OpenRoaming IDP Onboarding: Procedure for Identity Providers to Become a Member of the OpenRoaming Federation through RADSEC Leveraging Cisco Intermediate CA (WBA Root)*. Cisco.
- [17] Inc. Cisco Systems. 2024. *OpenRoaming Integration with Cisco Spaces*. <https://documentation.meraki.com> Last updated Jul 16, 2024. © 2024 Cisco Systems, Inc.
- [18] Cisco Systems, Inc. 2024. *Configuring RADSec (MR)*. Cisco Systems, Inc. [https://documentation.meraki.com/MR/Encryption\\_and\\_Authentication/MR\\_RADSec](https://documentation.meraki.com/MR/Encryption_and_Authentication/MR_RADSec) Last updated Oct 8, 2024.
- [19] Cisco Systems Inc. 2024. *Freeradius: Configure freeradius to work with EAP-TLS authentication*. Cisco Systems Inc. [https://documentation.meraki.com/MR/Encryption\\_and\\_Authentication/Freeradius%3A\\_Configure\\_freeradius\\_to\\_work\\_with\\_EAP-TLS\\_authentication](https://documentation.meraki.com/MR/Encryption_and_Authentication/Freeradius%3A_Configure_freeradius_to_work_with_EAP-TLS_authentication) Last updated Oct 5, 2020, © 2024 Cisco Systems, Inc.
- [20] Cisco Systems, Inc. 2024. Meraki Dashboard: Data for Marie Testing. <https://n560.meraki.com/Marie-network-wi/n/Kr1c-d4wb/manage/clients>
- [21] Cisco Systems, Inc. 2024. Meraki Homepage. <https://meraki.cisco.com/> © 2024 Cisco Systems, Inc.
- [22] Cisco Systems, Inc. 2024. What is a Wireless LAN? <https://www.cisco.com/site/us/en/learn/topics/security/what-is-a-wireless-lan.html#tabs-a107e9a621-item-6caff3e5bb-tab> Accessed: 2024-10-31.
- [23] Cisco Systems, Inc., Meraki. 2024. Hotspot 2.0. [https://documentation.meraki.com/MR/Other\\_Topics/Hotspot\\_2.0](https://documentation.meraki.com/MR/Other_Topics/Hotspot_2.0) Last updated: 2024-10-03, Accessed: 2024-10-31.

- [24] Cisco Systems, Inc., Meraki. 2024. Hotspot 2.0 Configuration Example. [https://documentation.meraki.com/MR/Other\\_Topics/Hotspot\\_2.0\\_Configuration\\_Example](https://documentation.meraki.com/MR/Other_Topics/Hotspot_2.0_Configuration_Example) Last updated: 2024-03-22, Accessed: 2024-10-31.
- [25] Google Cloud. 2024. *Identity Platform documentation*. [https://cloud.google.com/security/products/identity-platform?hl=fr&\\_gl=1\\*13pqsse\\*\\_ga\\*MTUzMTMwMTU5OS4xNzI4ODQ3NTM2\\*\\_ga\\_WH2QY8WWF5\\*MTczMTk00TAzMS4xOC4xLjE3MzE5NTAwNTUuNTIuMC4w](https://cloud.google.com/security/products/identity-platform?hl=fr&_gl=1*13pqsse*_ga*MTUzMTMwMTU5OS4xNzI4ODQ3NTM2*_ga_WH2QY8WWF5*MTczMTk00TAzMS4xOC4xLjE3MzE5NTAwNTUuNTIuMC4w) © 2024 Google Cloud.
- [26] VMware End-User Computing. 2020. *OAuth 2.0 and OpenID Connect (OIDC): Technical Overview*. <https://www.youtube.com/watch?v=rTz1F-U9Y6Y> 166,695 views.
- [27] Arran Cudbard-Bell. 2019. *Answer to: FreeRADIUS request without User-Password attribute for REST module*. <https://stackoverflow.com/questions/57648427/freeradius-request-without-user-password-attribute-for-rest-module/57662829#57662829> Stack Overflow, Answered Aug 26, 2019. Copyright 2018, The FreeRADIUS Project, licensed under CC BY.
- [28] Curity. 2023. *Claims Explained*. <https://curity.io/resources/learn/what-are-claims-and-how-they-are-used/> This article has been updated on: 2023-02-21, Published by Curity, 7 min read.
- [29] DB-Engines. 2024. DB-Engines Ranking of Key-value Stores. <https://db-engines.com/en/ranking/key-value+store> Accessed: 2024-11-12, Copyright © 2024 Red Gate Software Ltd.
- [30] DB-Engines. 2024. System Properties Comparison Redis vs. RocksDB. <https://db-engines.com/en/system/Redis%3BRocksDB> Copyright © 2024 Red Gate Software Ltd.
- [31] DeepL. 2024. *Better writing with DeepL Write*. <https://www.deepl.com/en/write> Accessed: 2024-10-07.
- [32] Google Developers. 2024. *Firebase Console*. <https://console.firebase.google.com/project> © 2024 Google Developers.
- [33] Google Developers. 2024. *Meet Android Studio*. <https://developer.android.com/studio/intro> © 2024 Google Developers.
- [34] Google Developers. 2024. Using OAuth 2.0 to Access Google APIs. <https://developers.google.com/identity/protocols/oauth2> Accessed: 2024-11-12.
- [35] DNASpaces. 2019. openroaming-auth-service. <https://www.in-github.cisco.com/DNASpaces/openroaming-auth-service>. Accessed: 11-09-2024, first commit by Nikhil Gupta (nigupta2).

- [36] Benoit Donnet. 2023-2024. Introduction to Computer Security, Part 1: Cryptography, Chapter 3: Symmetric Cryptography, Chapter 4: Asymmetric Cryptography and Chapter 6: Key Distribution; Part 2: Networking, Chapter 5: Network Attacks. INFO0045 - ULiège - Academic Year 2023/2024.
- [37] Dragonfly. [n.d.]. Top 26 Key-Value Databases Compared. <https://www.dragonflydb.io/guides/key-value-databases> Accessed: 2024-11-12.
- [38] eID PKI. 2024. eID PKI Repository Homepage. <https://repository.eidpki.belgium.be/#/home> Accessed: 2024-11-05.
- [39] Entrust.com. 2024. *A Quick Guide to eIDAS, Electronic Signatures, and Digital Certificates*. <https://www.entrust.com/sites/default/files/documentation/ebook/eidas-digital-signing-guide-eb.pdf> Accessed: 2024-10-07, ©2023 Entrust Corporation.
- [40] Dhruva Borthakur et al. 2023. RocksDB Overview. <https://github.com/facebook/rocksdb/wiki/RocksDB-Overview> © 2024 GitHub, Inc.
- [41] European Commission. 2024. eID. <https://ec.europa.eu/digital-building-blocks/sites/display/DIGITAL/eID> Accessed: 2024-11-04.
- [42] FOD Beleid en Ondersteuning – DG Digitale Transformatie. 2017. *e-ID Official Website*. <https://eid.belgium.be/en> Powered by FOD Beleid en Ondersteuning – DG Digitale Transformatie | 11.0.0 © 2017 CSAM.
- [43] Dr. Warwick S. Ford, Dr. Santosh Chokhani, Stephen S. Wu, Randy V. Sabett, and Charles (Chas) R. Merrill. 2003. Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework. RFC 3647. <https://doi.org/10.17487/RFC3647>
- [44] FPS BOSA. 2024. *FAS-OIDC Integration Guide*. <https://bosa.belgium.be/sites/default/files/content/documents/FAS%2520OIDC%2520-%2520Integration%2520Guide.pdf> version 6.5.
- [45] Paul Funk and Simon Blake-Wilson. 2008. Extensible Authentication Protocol Tunneled Transport Layer Security Authenticated Protocol Version 0 (EAP-TTLSv0). RFC 5281. <https://doi.org/10.17487/RFC5281>
- [46] Michal Garcarz and Thomas Wall. 2024. *Understand EAP-FAST and Chaining Implementations on AnyConnect NAM and ISE*. Technical Report. Cisco Systems, Inc. <https://www.cisco.com/c/en/us/support/docs/wireless-mobility/eap-fast/200322-Understanding-EAP-FAST-and-Chaining-imp.html> Document ID: 200322, Last updated: 2016-05-20, Accessed: 10/10/2024.

- [47] geeksforgeeks. [n. d.]. Difference between Cassandra and Redis. *GeeksforGeeks* ([n. d.]). <https://www.geeksforgeeks.org/difference-between-cassandra-and-redis/> Last updated: 13 Jul, 2020.
- [48] geeksforgeeks. 2024. Advanced Encryption Standard (AES). <https://www.geeksforgeeks.org/advanced-encryption-standard-aes/> Last updated: 2024-07-16, Accessed: 2024-10-31, @GeeksforGeeks, Sanchhaya Education Private Limited, All rights reserved.
- [49] GeeksforGeeks. 2024. *IEEE 802.11 Architecture*. <https://www.geeksforgeeks.org/ieee-802-11-architecture/> Last Updated: 06 Mar, 2024. Sanchhaya Education Private Limited, All rights reserved.
- [50] Google Developers. 2024. *Firebase Cloud Messaging*. Firebase. <https://firebase.google.com/docs/cloud-messaging/> licensed under the Creative Commons Attribution 4.0 License, and code samples are licensed under the Apache 2.0 License.
- [51] Google Developers. 2024. Firebase documentation. <https://firebase.google.com/docs> Accessed: 2024-11-12.
- [52] Google Developers. 2024. *Wi-Fi suggestion API for internet connectivity*. <https://developer.android.com/develop/connectivity/wifi/wifi-suggest?hl=fr>
- [53] Google for Developers. 2024. Android's Kotlin-first Approach. <https://developer.android.com/kotlin/first> Accessed: 2024-11-12.
- [54] Google for Developers. 2024. *WifiManager*. Android. <https://developer.android.com/reference/kotlin/android/net/wifi/WifiManager> Content and code samples on this page are subject to the licenses described in the Content License. Java and OpenJDK are trademarks or registered trademarks of Oracle and/or its affiliates. Last updated 2024-12-18 UTC.
- [55] Patrick Grubbs. 2024. What is RadSec? <https://www.securew2.com/blog/what-is-radsec> Accessed: 2024-11-03.
- [56] Arnt Gulbrandsen and Dr. Levon Esibov. 2000. A DNS RR for specifying the location of services (DNS SRV). RFC 2782. <https://doi.org/10.17487/RFC2782>
- [57] Dick Hardt. 2012. The OAuth 2.0 Authorization Framework. RFC 6749. <https://doi.org/10.17487/RFC6749>
- [58] Anusha Harish. 2024. An Overview Of Passpoint In Network Infrastructure. <https://www.securew2.com/blog/what-is-passpoint> Accessed: 2024-11-03.
- [59] Anusha Harish. 2024. *How To Protect Your Network From Wi-Fi Spoofing Attacks?* © Copyright 2024 Cloud RADIUS. <https://www.cloudradius.com/wi-fi-spoofing-a-major-threat-to-network-security/>



- [60] Derrick Harris. 2013. Facebook's latest open source effort: a flash-powered database called RocksDB. <https://web.archive.org/web/20200224065917/https://gigaom.com/2013/11/21/facebook-latest-open-source-effort-a-flash-powered-database-called-rocksdb/> 2020 GigaOm All Rights Reserved.
- [61] Hughes Systique Corporation. 2022. *Demystifying Wi-Fi OpenRoaming*. <https://www.hsc.com/resources/blog/demystifying-wi-fi-openroaming/>
- [62] Cisco Systems Inc. 2024. *bridge it home*. <https://onesearch.cisco.com/ciscoit/chatgpt/home> Copyright © 2024 Cisco Systems Inc. All rights reserved.
- [63] JetBrains. 2024. Kotlinx Serialization JSON: Kotlin Multiplatform Serialization Runtime Library. <https://mvnrepository.com/artifact/org.jetbrains.kotlinx/kotlinx-serialization-json-jvm> License: Apache 2.0, Accessed: 2024-11-12.
- [64] Michael B. Jones, John Bradley, and Nat Sakimura. 2015. JSON Web Token (JWT). RFC 7519. <https://doi.org/10.17487/RFC7519>
- [65] Michael B. Jones and Dick Hardt. 2012. The OAuth 2.0 Authorization Framework: Bearer Token Usage. RFC 6750. <https://doi.org/10.17487/RFC6750>
- [66] Simon Josefsson. 2006. The Base16, Base32, and Base64 Data Encodings. RFC 4648. <https://doi.org/10.17487/RFC4648>
- [67] Krishna Sankar, Sri Sundaralingam, Darrin Miller, Andrew Balinsky. 2005. *Cisco Wireless LAN Security* (1st ed.). Cisco Press, Pearson Education, Hoboken, NJ. Part of the Networking Technology series.
- [68] Cornelius Kölbel. 2024. Configuration of rlm\_rest. [https://privacyidea.readthedocs.io/en/latest/application\\_plugins/rlm\\_rest.html](https://privacyidea.readthedocs.io/en/latest/application_plugins/rlm_rest.html) Copyright 2014-2024, privacyIDEA, Created using Sphinx 4.5.0.
- [69] Guy Leduc. 2022. Securing Networks, Chapter 4: Securing TCP Connections. Course materials from Liege University. Based on content from "Computer Networking: A Top-Down Approach" 7th edition by Kurose and Ross, Addison-Wesley, 2016 (section 8.6).
- [70] Guy Leduc. 2023. Network Infrastructure, Part 3, Chapter 1 and 2. Course materials from Liege University. Based on content from Chapter 7 of "Computer Networking: A Top-Down Approach" 8th edition by Kurose and Ross, Pearson, 2020.
- [71] Sébastien Marchal. 2024. Next-Gen Onboarding TDM. (February 2024). Cisco confidential. Cisco and its affiliates.
- [72] Michael H. Mealling and Dr. Ron Daniel. 2000. The Naming Authority Pointer (NAPTR) DNS Resource Record. RFC 2915. <https://doi.org/10.17487/RFC2915>

- [73] Meta Open Source. 2022. RocksDB. <https://rocksdb.org/> Copyright © 2022 Meta Platforms, Inc..
- [74] Microsoft. 2024. What is: Multifactor Authentication. <https://support.microsoft.com/en-us/topic/what-is-multifactor-authentication-e5e39437-121c-be60-d123-eda06bddf661> Accessed: 2024-11-04.
- [75] Maria Milona. [n. d.]. Meraki • etymology • Greek word. <https://www.pinterest.com/pin/774124927954062/> taken from Pinterest.
- [76] Kathleen Moriarty, Burt Kaliski, Jakob Jonsson, and Andreas Rusch. 2016. PKCS #1: RSA Cryptography Specifications Version 2.2. RFC 8017. <https://doi.org/10.17487/RFC8017>
- [77] JetBrains mvnrepository. 2024. Adding Ktor Dependencies. <https://ktor.io/docs/server-dependencies.html> Last modified: 02 April 2024, Accessed: 2024-11-12, Copyright © 2000-2024 JetBrains s.r.o.
- [78] JetBrains mvnrepository. 2024. Kotlin Standard Library. <https://mvnrepository.com/artifact/org.jetbrains.kotlin/kotlin-stdlib> License: Apache 2.0, Accessed: 2024-11-12.
- [79] JetBrains mvnrepository. 2024. Kotlin Test JUnit: Kotlin Test Library Support for JUnit. <https://mvnrepository.com/artifact/org.jetbrains.kotlin/kotlin-test-junit> License: Apache 2.0, Accessed: 2024-11-12.
- [80] JetBrains mvnrepository. 2024. Logback Classic Module: Implementation of the SLF4J API for Logback. <https://mvnrepository.com/artifact/ch.qos.logback/logback-classic> License: EPL 1.0, LGPL 2.1, Accessed: 2024-11-12.
- [81] JetBrains mvnrepository. 2024. OkHttp: Square’s Meticulous HTTP Client for Java and Kotlin. <https://mvnrepository.com/artifact/com.squareup.okhttp3/okhttp> License: Apache 2.0, Accessed: 2024-11-12.
- [82] JetBrains mvnrepository. 2024. RocksDB JNI: RocksDB Fat Jar with Platform-Specific Libraries. <https://mvnrepository.com/artifact/org.rocksdb/rocksdbjni> License: Apache 2.0 and GPL 2.0, Accessed: 2024-11-12.
- [83] JetBrains mvnrepository. 2024. SLF4J API Module: API for SLF4J (The Simple Logging Facade for Java). <https://mvnrepository.com/artifact/org.slf4j/slf4j-api> License: MIT, Accessed: 2024-11-12.
- [84] MyPension. [n. d.]. MyPension. <https://www.mypension.be/en>. Accessed: 11/09/2024.

- [85] eIDAS Network and Walter Arrighetti. 2021. *Guidance for the application of the levels of assurance which support the eIDAS Regulation*.
- [86] NetworkLessons.com. 2024. *EAPOL (Extensible Authentication Protocol over LAN)*. <https://networklessons.com/cisco/ccnp-encor-350-401/eapol-extensible-authentication-protocol-over-lan> CCNP ENCOR 350-401, © 2013 - 2024 NetworkLessons.com.
- [87] NetworkRADIUS. 2021. How Authentication Protocols Work: PAP, CHAP, MS-CHAP, and EAP. <https://www.networkradius.com/articles/2022/02/20/how-authentication-protocols-work.html> Accessed: 2024-10-31.
- [88] Henrik Nielsen, Jeffrey Mogul, Larry M Masinter, Roy T. Fielding, Jim Gettys, Paul J. Leach, and Tim Berners-Lee. 1999. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616. <https://doi.org/10.17487/RFC2616>
- [89] Okta, Inc. 2024. *What is OAuth 2.0?* <https://auth0.com/intro-to-iam/what-is-oauth-2> Accessed: 2024-11-05, © 2024 Okta, Inc. All Rights Reserved.
- [90] OpenID Foundation. 2024. *What is OpenID Connect*. <https://openid.net/developers/how-connect-works/> Accessed: 2024-10-07.
- [91] Ashwin Palekar, Simon Josefsson, Daniel Simon, and Glen Zorn. 2004. *Protected EAP Protocol (PEAP) Version 2*. Internet-Draft draft-josefsson-pppext-eap-tls-eap-10. Internet Engineering Task Force. <https://datatracker.ietf.org/doc/draft-josefsson-pppext-eap-tls-eap/10/> Work in Progress.
- [92] Tommy Pauly, Steven Valdez, and Christopher A. Wood. 2024. The Privacy Pass HTTP Authentication Scheme. RFC 9577. <https://doi.org/10.17487/RFC9577>
- [93] Vivek Raj. 2024. *EAP-TLS vs. EAP-TTLS/PAP*. <https://www.securew2.com/blog/eap-tls-vs-eap-ttls-pap> SecureW2.
- [94] Eytan Raphaely. 2024. EAP-TLS vs. PEAP-MSCHAPv2: Which Authentication Protocol is Superior? <https://www.securew2.com/blog/eap-tls-vs-peap-mschap2-which-authentication-protocol-is-superior> Accessed: 2024-10-31.
- [95] Eric Rescorla and Tim Dierks. 2008. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246. <https://doi.org/10.17487/RFC5246>
- [96] Allan Rubens, Carl Rigney, Steve Willens, and William A. Simpson. 2000. Remote Authentication Dial In User Service (RADIUS). RFC 2865. <https://doi.org/10.17487/RFC2865>

- [97] Vincent Ryan. 2014. *JEP 229: Create PKCS12 Keystores by Default*. <https://openjdk.org/jeps/229> © 2024 Oracle Corporation and/or its affiliates. License: GPLv2, Updated: 2018/01/11.
- [98] Joseph A. Salowey, Hao Zhou, Nancy Cam-Winget, and David McGrew. 2007. The Flexible Authentication via Secure Tunneling Extensible Authentication Protocol Method (EAP-FAST). RFC 4851. <https://doi.org/10.17487/RFC4851>
- [99] Paresh Sawant and Bart Brinckman. 2024. *Extensible Authentication Protocol (EAP) Using Privacy Pass Token*. Internet-Draft draft-sawant-eap-ppt-01. Internet Engineering Task Force. <https://datatracker.ietf.org/doc/draft-sawant-eap-ppt/01/> Work in Progress.
- [100] SecureNet. 2023. *25 802 1x and EAP Concepts*. <https://www.youtube.com/watch?v=dwPipQscMjM> 11,762 views, INDIA.
- [101] SecureNet. 2023. *27 WPA2 and 802 11i Concepts*. <https://www.youtube.com/watch?v=1GKholXaPls> 1,505 views, INDIA.
- [102] Service Général d’Informatique, Université de Liège. 2024. Réseau et WiFi. [https://www.campus.uliege.be/cms/c\\_11230598/fr/reseau-et-wifi](https://www.campus.uliege.be/cms/c_11230598/fr/reseau-et-wifi) Accessed: 2024-10-31, © Copyright ULiège 2023.
- [103] Shubhanjaytiwari. 2022. Key-Value Data Model in NoSQL. [https://www.geeksforgeeks.org/user/shubhanjaytiwari/contributions/?itm\\_source=geeksforgeeks&itm\\_medium=article\\_author&itm\\_campaign=auth\\_user](https://www.geeksforgeeks.org/user/shubhanjaytiwari/contributions/?itm_source=geeksforgeeks&itm_medium=article_author&itm_campaign=auth_user). Last Updated: 17 Feb, 2022.
- [104] Daniel Simon, Ryan Hurst, and Dr. Bernard D. Aboba. 2008. The EAP-TLS Authentication Protocol. RFC 5216. <https://doi.org/10.17487/RFC5216>
- [105] William A. Simpson. 1992. PPP Authentication Protocols. RFC 1334. <https://doi.org/10.17487/RFC1334>
- [106] William A. Simpson. 1992. The Point-to-Point Protocol (PPP) for the Transmission of Multi-protocol Datagrams over Point-to-Point Links. RFC 1331. <https://doi.org/10.17487/RFC1331>
- [107] Camryn Smith. 2023. *Is Public Wi-Fi Safe? No, but It Is Necessary*. <https://www.allconnect.com/blog/is-public-wifi-safe> Copyright © 2024 Allconnect. A Red Ventures Company. All rights reserved..
- [108] SPF BOSA. 2024. Federal Authentication Service (FAS). <https://bosa.belgium.be/fr/services/federal-authentication-service-fas> Accessed: 2024-11-04.
- [109] SPF Stratégie Appui, BOSA. 2024. *MyGov Belgium*. <https://mygov.be/> © BOSA - SPF Stratégie Appui.

- [110] StackShare. 2024. Redis vs RocksDB. <https://stackshare.io/stackups/redis-vs-rocksdb> Copyright © 2024 FOSSA, Inc. All rights reserved..
- [111] Sunny Classroom. 2019. AAA Framework and RADIUS. <https://www.youtube.com/watch?v=feHpDc1cLXM> Accessed: 2024-11-3, YouTube video, 145,885 views.
- [112] Telecom Trainer. 2024. ANQP (Access Network Query Protocol). <https://www.telecomtrainer.com/anqp-access-network-query-protocol/> Last updated: 2023-02-25, Accessed: 2024-10-31.
- [113] Johan Terve. 2024. *All You Need To Know About OpenRoaming*. Technical Report. Enea. <https://info.enea.com/openroaming-white-paper> White Paper, Published: 2024-01-16, Accessed: 2024-10-31.
- [114] The FreeRADIUS Server Project and Contributors. 2023. FreeRADIUS: Documentation. <https://www.freeradius.org/documentation/> Accessed: 2024-11-14.
- [115] The FreeRADIUS Server Project and Contributors. 2023. FreeRADIUS: rlm\_eap. <https://networkradius.com/doc/current/raddb/mods-available/eap.html> Accessed: 2024-11-14.
- [116] The FreeRADIUS Server Project and Contributors. 2023. FreeRADIUS: rlm\_eap\_tls. <https://networkradius.com/doc/current/raddb/mods-available/eap/tls.html> Accessed: 2024-11-14.
- [117] The FreeRADIUS Server Project and Contributors. 2023. FreeRADIUS: rlm\_rest. <https://networkradius.com/doc/current/raddb/mods-available/rest.html> Accessed: 2024-11-14.
- [118] The FreeRADIUS Server Project and Contributors. 2023. FreeRADIUS: We Authenticate the Internet. <https://www.freeradius.org/> Accessed: 2024-11-12.
- [119] VOCAL Technologies. 2024. EAPoL Protocol – Extensible Authentication Protocol over LAN. <https://vocal.com/secure-communication/eapol-extensible-authentication-protocol-over-lan/> Accessed: 2024-10-31.
- [120] John Vollbrecht, James D. Carlson, Larry Blunk, Dr. Bernard D. Aboba, and Henrik Levkowetz. 2004. Extensible Authentication Protocol (EAP). RFC 3748. <https://doi.org/10.17487/RFC3748>
- [121] Radhika Vyas. 2024. *LEAP Authentication and How It Works*. <https://www.securew2.com/blog/leap-authentication-and-how-it-works> © 2024 SecureW2.
- [122] Wi-Fi Alliance. 2019. *Hotspot 2.0 Specification, Version 3.1*. Technical Report. Wi-Fi Alliance. <https://www.wi-fi.org/file/>

- hotspot-20-specification-version-31 WI-FI ALLIANCE PROPRIETARY – SUBJECT TO CHANGE WITHOUT NOTICE. Used with the permission of Wi-Fi Alliance under the terms as stated in this document.
- [123] Wi-Fi Alliance. 2021. *Wi-Fi® global economic value to reach \$5 trillion in 2025*. Technical Report. <https://www.wi-fi.org/news-events/newsroom/wi-fi-global-economic-value-to-reach-5-trillion-in-2025> study from Telecom Advisory Services (TAS), headed by Dr. Raul Katz.
  - [124] Wi-Fi Alliance. 2024. Passpoint. <https://www.wi-fi.org/discover-wi-fi/passpoint> Accessed: 2024-10-31.
  - [125] Wi-Fi Alliance. 2024. *Wi-Fi®*. <https://www.wi-fi.org/> © 2024 Wi-Fi Alliance. All rights reserved.
  - [126] Klaas Wierenga, Mike McCauley, Stefan Winter, and Stig Venaas. 2012. Transport Layer Security (TLS) Encryption for RADIUS. RFC 6614. <https://doi.org/10.17487/RFC6614>
  - [127] Klaas Wierenga, Stefan Winter, and Tomasz Wolniewicz. 2015. The eduroam Architecture for Network Roaming. RFC 7593. <https://doi.org/10.17487/RFC7593>
  - [128] Wikipedia contributors. 2024. CAPTCHA — Wikipedia, The Free Encyclopedia. <https://en.wikipedia.org/w/index.php?title=CAPTCHA&oldid=1256528649>. [Online; accessed 29-November-2024].
  - [129] Wikipedia contributors. 2024. Fully qualified domain name — Wikipedia, The Free Encyclopedia. [https://en.wikipedia.org/w/index.php?title=Fully\\_qualified\\_domain\\_name&oldid=1254798503](https://en.wikipedia.org/w/index.php?title=Fully_qualified_domain_name&oldid=1254798503). [Online; accessed 1-November-2024].
  - [130] Wikipedia contributors. 2024. IEEE 802.11 — Wikipedia, The Free Encyclopedia. [https://en.wikipedia.org/w/index.php?title=IEEE\\_802.11&oldid=1249819320](https://en.wikipedia.org/w/index.php?title=IEEE_802.11&oldid=1249819320) [Online; accessed 10-October-2024].
  - [131] Wikipedia contributors. 2024. Quality of service — Wikipedia, The Free Encyclopedia. [https://en.wikipedia.org/w/index.php?title=Quality\\_of\\_service&oldid=1253977811](https://en.wikipedia.org/w/index.php?title=Quality_of_service&oldid=1253977811). [Online; accessed 1-November-2024].
  - [132] Wikipedia contributors. 2024. Temporal Key Integrity Protocol — Wikipedia, The Free Encyclopedia. [https://en.wikipedia.org/w/index.php?title=Temporal\\_Key\\_Integrity\\_Protocol&oldid=1250223092](https://en.wikipedia.org/w/index.php?title=Temporal_Key_Integrity_Protocol&oldid=1250223092) [Online; accessed 31-October-2024].
  - [133] Wikipedia contributors. 2024. Wi-Fi Protected Access — Wikipedia, The Free Encyclopedia. [https://en.wikipedia.org/w/index.php?title=Wi-Fi\\_Protected\\_Access&oldid=1252837652](https://en.wikipedia.org/w/index.php?title=Wi-Fi_Protected_Access&oldid=1252837652). [Online; accessed 31-October-2024].

- [134] Stefan Winter and Mike McCauley. 2015. Dynamic Peer Discovery for RADIUS/TLS and RADIUS/DTLS Based on the Network Access Identifier (NAI). RFC 7585. <https://doi.org/10.17487/RFC7585>
- [135] Wireless Broadband Alliance. 2023. *OpenRoaming*. <https://wballiance.com/openroaming/> Accessed: 2023-10-05, Wireless Broadband Alliance Inc. © 2024. All rights reserved. Designed Built by Fresh01.
- [136] Inc. Wireless Broadband Alliance, Cisco Systems, Intel Corporation, SingleDigits, and Cisco Systems. 2024. *WBA OpenRoaming Wireless Federation*. Internet-Draft draft-tomas-openroaming-03. Internet Engineering Task Force. <https://www.ietf.org/archive/id/draft-tomas-openroaming-03.html> Expires: 26 January 2025.
- [137] Shankar Ayyamperumal wireless-cnt adm. 2020. *Cisco OpenRoaming Whitepaper*. Technical Report. Business Unit, Cisco. <https://www.cisco.com/c/en/us/solutions/collateral/enterprise-networks/dna-spaces/white-paper-c11-742106.html> Audience: All Employees, All Partners, All Distributors, Gold Partners, Premier Partners, Select Partners..
- [138] Glen Zorn. 2000. Microsoft PPP CHAP Extensions, Version 2. RFC 2759. <https://doi.org/10.17487/RFC2759>
- [139] Glen Zorn and Steve Cobb. 1998. Microsoft PPP CHAP Extensions. RFC 2433. <https://doi.org/10.17487/RFC2433>