



**Nuno Morais
Marques**

**Integração do paradigma de Captive Portals com a
arquitetura 802.1X**

**Captive Portal paradigm integration with 802.1X
architecture**



**Nuno Morais
Marques**

**Integração do paradigma de Captive Portals com a
arquitetura 802.1X**

**Captive Portal paradigm integration with 802.1X
architecture**

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor André Ventura da Cruz Marnoto Zúquete, Professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor João Paulo da Silva Barraca, Professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president

Prof. Doutora Susana Isabel Barreto de Miranda Sargento
professora associada c/ agregação da Universidade de Aveiro

vogais / examiners committee

Prof. Doutor Manuel Eduardo Carvalho Duarte Correia
professor auxiliar da Faculdade de Ciências da Universidade do Porto

Prof. Doutor André Ventura da Cruz Marnoto Zúquete
professor auxiliar da Universidade de Aveiro

agradecimentos / acknowledgements

Em toda esta dissertação, este é o tópico que mais me deixa à vontade por conseguir dizer e agradecer a todas as pessoas que sempre se preocuparam e me apoiaram em todo este percurso. Primeiro que tudo, quero agradecer aos meus pais, Maria Helena Vieira Morais e Nuno Cabral Martins Marques, por me terem criado com tanta dedicação e tanto esforço e por sempre me perguntarem como estavam as coisas na universidade, em todas as alturas que se encontravam comigo. Agradeço também à minha namorada sempre bastante preocupada e atenciosa, Keitellyne Manuella Pereira Bahia, que, embora não percebesse nada do assunto, sempre tentava saber sobre do que se tratava esta dissertação e que sempre esteve lá nos momentos mais desmotivantes. Quero também agradecer aos meus grandes amigos de curso, Bruno Alves, André Santos, Ivo Silva, Rui Lebre, Ricardo Martins e Tiago Henriques por sempre me acompanharem em todo este percurso universitário e por me terem proporcionado momentos que sempre irei guardar. Ainda um especial agradecimento aos meus grandes amigos André Lemos Marques e Miguel António Azevedo Viana de Oliveira, que apesar de não lhes pertencer esta dissertação sempre se preocuparam com o estado dela. Em tom de repetição, também agradeço a todos os meus amigos da Associação BEST Aveiro e à restante direção da mesma pertencente ao meu mandato (Orlando Pinheiro, Andreia Santos, Joana Caneco e Filipe Ferreira) que tantas frustrações partilharam comigo, que tanto se preocuparam e que tantas alegrias me deram ao longo deste longo ano (passo a redundância). Por fim, agradeço a todos os restantes, família e amigos, pois não me posso alongar, contudo ninguém foi esquecido.

Palavras Chave

802.1X, EAP, TLS, segurança, redes informáticas, hotspots, captive portals.

Resumo

Num cenário em que as redes *hotspot* estão a ser progressivamente mais usadas e presentes e a obter mais subscritores, com a quantidade de informação sensível que neste tipo de redes é transmitida e com a variedade destes mesmos utilizadores que podem ser ou não de confiança, são necessários mecanismos de segurança que garantam a confidencialidade e integridade de dados, assim como garantir que redes anunciadas sejam autenticadas, evitando redes malignas. Os *captive portals*, portais providenciados por redes deste tipo onde se efetua *log in*, são ainda um maior risco pois implicam a transmissão de dados sensíveis de maneira não *standard*. Este trabalho explora as fraquezas deste paradigma e apresenta uma solução que pretende colmatá-las, baseada na arquitetura 802.1X. Esta solução passa por criar uma extensão do protocolo EAP a fim de poder integrar a autenticação via HTTP com o processo de autenticação do 802.1X.

Keywords

802.1X, EAP, TLS, security, networks, hotspots, captive portals.

Abstract

In a scenario where hotspot networks are increasingly being used, present and obtaining more subscribers, with the amount of sensitive information exchanged on this type of networks and with the variety of their users, which may not be trustworthy, there is a need of implementing security mechanisms that guarantee data confidentiality and integrity, as well as to guarantee that announced networks are genuine, avoiding rogue networks. Captive portals are portals provided by networks of this type where a user logs in; they are a greater risk as they imply the transmission of sensitive data on a non-standardized way. This work explores the weaknesses of this paradigm and describes a solution that intends to suppress them, based on the 802.1X architecture. This solution consists on creating an EAP-compliant protocol in order to integrate an HTTP-based authentication within the 802.1X authentication framework.

Contents

Contents	i
List of Figures	iii
List of Tables	v
Glossary	vii
1 Introduction	1
1.1 Motivation	1
1.2 Objectives	2
1.3 Thesis Structure	3
2 Context	5
2.1 Extensible Authentication Protocol	5
2.1.1 EAP: Packet Structure	6
2.1.2 EAP: Methods	7
2.1.3 EAP: Generic EAP Conversation	8
2.2 IEEE 802.1X	10
2.2.1 802.1X: Entities	10
2.2.2 802.1X: Controlled and Uncontrolled Ports	10
2.2.3 802.1X: Authentication Process	11
2.2.4 How Linux uses 802.1X	12
2.3 Captive Portals	14
2.3.1 Implementations	14
2.3.2 Security Limitations	15
2.3.3 Captive Portal Engines	15
2.3.4 ChilliSpot	16
3 Related Work	21
3.1 Secure MAC-layer Protocol for Captive Portals	21
3.2 Hotspot 2.0	22
3.3 Apple’s Captive Portal Assistant	22
4 Architecture	23
4.1 Overview	23
4.2 Extensible Authentication Protocol for Secure Hotspots	24
4.2.1 EAP-SH: Message Structure	25
4.2.2 EAP-SH: Flowchart	27

4.2.3	EAP-SH: Certificate Infrastructure	29
4.2.4	EAP-SH: Detailed Operation	32
5	Implementation	39
5.1	WPA Supplicant	39
5.1.1	Configuration	41
5.1.2	Data structures	42
5.1.3	EAP-SH Supplicant States	44
5.1.4	Fragmentation and Defragmentation	47
5.2	RADIUS	48
5.2.1	Configuration	48
5.2.2	Data Structures	49
5.2.3	EAP-SH Server States	51
5.2.4	HTTP Server	54
6	Security Analysis	55
6.1	Dolev-Yao Model Conditions	55
6.1.1	Rogue Access Points	56
6.1.2	Client Certificate Owner	56
6.1.3	Server Certificate Owner	56
6.1.4	Client Identity	56
6.1.5	Captive Portal Circumvention	57
6.1.6	Captive Portal Impersonation	57
7	Conclusion	59
	References	61

List of Figures

2.1	EAP Packet structure	6
2.2	Example of an EAP Conversation	9
2.3	802.1X Basic Architecture	10
2.4	Example of an 802.1X Authentication Process	12
2.5	ChilliSpot Architecture	17
2.6	ChilliSpot Authentication Process	19
4.1	EAP-TLS Message Format	25
4.2	Modified EAP-TLS Flags byte for the first stage of the protocol	26
4.3	EAP-SH Message Format	26
4.4	EAP-SH Flags byte for the second stage of the protocol	27
4.5	EAP-SH Flowchart	28
4.6	EAP-SH Certification Hierarchy Model	31
4.7	EAP-SH Phase 1 Message Exchange - Success	32
4.8	EAP-SH Phase 1 Message Exchange - Failure	34
4.9	EAP-SH Phase 2 - HTTP Protocol Mechanism	35
4.10	EAP-SH Phase 2 Message Exchange - Captive Portal Login Success	36
4.11	EAP-SH Phase 2 Message Exchange - Certificate Signing and Install	37
5.1	EAP-SH: Supplicant State Machine	46
5.2	EAP-SH: RADIUS State Machine	53
5.3	Implemented Captive Portal	54

List of Tables

2.1	EAP Method Comparison	8
-----	---------------------------------	---

Glossary

AP	Access Point	LAN	Local Area Network
BSS	Basic Service Set	LDAP	Lightweight Directory Access Protocol
CA	Certification Authority	MAC	Medium Access Control
CHAP	Challenge-Handshake Authentication Protocol	MSK	Master Session Key
CRL	Certificate Revocation List	MTU	Maximum Transmission Unit
CSR	Certificate Signing Request	Nak	Not Acknowledged
DHCP	Dynamic Host Configuration Protocol	OSCP	Online Certificate Status Protocol
DNS	Domain Name System	OSA	Open System Authentication
EAP	Extensible Authentication Protocol	PKG	Private Key Generator
EAPoL	EAP over LAN	PPP	Point-to-Point Protocol
EAPoW	EAP over Wireless	PSK	Pre-Shared Key
GUI	Graphical User Interface	RADIUS	Remote Authentication Dial-In User Service
EAP-SH	Extensible Authentication Protocol for Secure Hotspots	SOHO	Small Office / Home Office
HTML	HyperText Markup Language	SSID	Service Set Identifier
HTTP	Hyper Text Transfer Protocol	TCP	Transmission Control Protocol
HTTPS	Hyper Text Transfer Protocol Secure	TLS	Transport Layer Security
ICMP	Internet Control Message Protocol	URL	Uniform Resource Locator
IP	Internet Protocol	WEP	Wired Equivalent Privacy
ISP	Internet Service Provider	WLAN	Wireless LAN
IEEE	Institute of Electrical and Electronics Engineers	WPA	Wi-Fi Protected Access
		WPA2	IEEE 802.11i

One

Introduction

This chapter intends to give a brief introduction to the thesis, describing the problem at stake and the objectives of this work.

'The way to get started is to quit talking and begin doing', Walt Disney

1.1 MOTIVATION

In the last century, developers and system designers always had the concern of developing new technologies to allow improvements on every other area. The truth is that it was quite successful and many other subjects benefited from this, for example, medicine and finances. The biggest problem, which was the least of concerns for who developed these systems, were the security considerations taken into account, which were almost none. As years passed by, new vulnerabilities were discovered and new systems considered such vulnerabilities when being designed and implemented.

Computer security had its origin in the military field. Nowadays, as technology advances, it also comes with a growing concern with security issues of existent and new network and data systems, in terms of some aspects:

1. Confidentiality, the property of making information not available to unauthorized entities;
2. Integrity, the property of maintaining and assuring the non-modification of information without authorization;
3. Availability, the property of making information available to legitimate users, when they request for it;
4. Authentication, the property of assuring the legitimacy of a user;
5. Non-repudiation, the property of assuring that a neutral third-party can be convinced that an event did or did not occur.

The Internet is no exception. The amount of sensitive information that everyday circulates through it is enormous, and it is tending to grow as time passes. Additionally,

not only transmitted information is at stake but also information stored in end-user's workstations and servers.

Public hotspots are networks of easy access, hence their usefulness. Also, their usage increases everyday, as well as the number of their subscribers. However, they are really dangerous when it comes to the transmission of sensitive information. This type of network usually comes with a plug-in, captive portals, which are web portals that allow a user to have access to the network as long as he has an account associated to the Internet Service Provider managing that network. On this case:

1. Clients traffic is not encrypted, and therefore can be captured in clear text by eavesdroppers;
2. Traffic to/from clients does not have any integrity control, and therefore man-in-the-middle attacks can be performed;
3. Clients and machines, together, can be fooled by rogue access points.
4. Clients can abuse the network through tunnels over protocols that are left open, such as DNS.

This thesis intends to solve those problems through the presented solution.

1.2 OBJECTIVES

This work's objective was to adapt the captive portals' paradigm to the 802.1X architecture, in order to solve the problems referred in section 1.1, namely traffic encryption, integrity control and mutual authentication. In detail, the objectives of this thesis are the following:

1. Comprehensive study to understand how captive portals work, and how authentication is performed on each of its many engines;
2. Study of the base principles of the EAP meta protocol, as well as how in Linux are EAP methods explored on the supplicant side, the client end of an 802.1X access architecture;
3. Design of a new EAP method to fit into the captive portal paradigm, to allow mutual authentication, preventing rogue access points and to allow client authentication, as well as to incorporate an encrypted tunnel to allow sensitive data to be protected and its integrity assured;

4. Development of a prototype in Linux that implements this EAP method design, both in the EAP client side (supplicant) and in the EAP server side (authentication server).

1.3 THESIS STRUCTURE

The remainder of this thesis is organized as follows. In chapter 2, the state of the art is presented, namely all EAP features, EAP packet structure, some EAP methods and its pros and cons and also a generic EAP conversation to show how EAP is processed. 802.1X is described in detail, its architecture, how it works and how does Linux make use of 802.1X on its supplicant side. Lastly, on the last section captive portals are explained, how they can be implemented and their security limitations, as well as some existing captive portal software access controllers and a case study of ChilliSpot, one of the captive portal engines.

In chapter 3, some related work is presented, which means solutions that already exist and somehow are related to this work. Firstly, a secure captive portal solution is going to be described, which uses a MAC-layer solution. Secondly, Hotspot 2.0, a new standard for public-access Wi-Fi that enables seamless roaming between Wi-Fi networks. Finally, the exploitation of a captive portal circumvention solution, Captive Portal Assistant, developed by Apple.

In chapter 4, the first pages present an overview of the solution and its purpose, followed by the detailed description of the solution distributed in five topics. The first topic overviews the protocol, and the next four topics describe its message structure, namely its fields and what are they for, a detailed flowchart about the whole authentication process, a detailed description of the certification infrastructure as well as a detailed description of the whole architecture's operation.

Chapter 5 presents the solution's implementation, describes the supplicant side and the server side implementations, and how they work when authentication is in progress.

Chapter 6 presents a security analysis based on Dolev-Yao Model conditions. It describes different scenarios in which an attacker would impersonate entities such as the client, the authenticator and the captive portal, and how the protocol defends itself

from these scenarios.

Finally, chapter 7 presents a conclusion about the discussed problem, the solution's description and advantages and future work perspectives.

Two

Context

This chapter intends to describe the state-of-the-art related with EAP, 802.1X and Captive Portals.

'The journey of a thousand miles begins with a single step', Lao Zi

2.1 EXTENSIBLE AUTHENTICATION PROTOCOL

The Extensible Authentication Protocol (EAP) [1] is a meta protocol designed to encapsulate authentication mechanisms and it is used in wireless networks (e.g. 802.1X [2]) and point-to-point connections (e.g. PPP [3]). This protocol does not have any security by itself, therefore the authentication method that it encapsulates must implement its own security. To this day, there are approximately 40 EAP authentication methods in use [4].

EAP has two main features: it provides a typed message exchange (there are only 4 types of messages, explained further below) and it provides extensibility, as said. The authentication methods are dynamically selected in a negotiation made between the authenticating peers, which is performed separately from any other action on the EAP data.

The requirements of EAP authentication methods for Wireless LANs, defined in RFC 4107 [5], are important to make the protocol have some security considerations in its designing, though not mandatory. These include:

1. Mutual authentication, which should be resistant to man-in-the-middle and dictionary attacks;
2. Guarantee the generation and distribution of a master secret key to both parties (supplicant and authenticator);
3. Protect the identity of supplicants.

In 802.1X, EAP is used to free the Access Point (AP), which is the authenticator, from

the burden of managing some of the particular aspects of the adopted authentication protocol, and so the authentication is managed by an authentication server. Using this methodology, it is possible to change the authentication method without changing the AP software. Such EAP-based feature made the use of several authentication methods in corporate networks possible, besides the common username-password model [4].

2.1.1 EAP: Packet Structure

All EAP messages have a common format, as described in RFC 3748 [1]. The figure below represents the packet structure of EAP messages. The fields are transmitted from left to right.

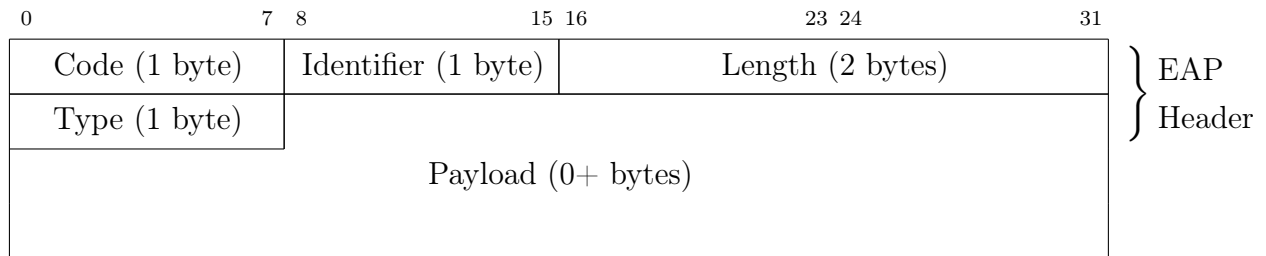


Figure 2.1: EAP Packet structure

The Code field has a size of one byte and represents the type of the EAP packet. EAP Codes are assigned as follows:

- Code 1 - EAP Request : This type of packet is sent by the authenticator to the supplicant. Request packets (with the same Identifier field) are sent until a valid EAP Response or a lower layer failure is received. Optionally, a retry counter can be configured;
- Code 2 - EAP Response : This type of packet is sent by the supplicant to the authenticator in reply to an EAP Request. The Response matches the Request Identifier field which it replies to;
- Code 3 - EAP Success : This type of packet is sent by the authenticator to the supplicant, after completing the authentication mechanism in progress, to indicate the first has done a successful authentication to the latter. This type of packet contains no Payload field.
- Code 4 - EAP Failure : This type of packet is sent by the authenticator to the supplicant, after completing the authentication protocol in progress, to indicate an unsuccessful authentication. This type of packet contains no Payload field.

The Identifier field has a size of one byte and is unique for each pair Request-Response, for the whole duration of the EAP conversation. This means that the EAP Response Identifier field is always the same as the current Request's. If they don't match, the authenticator discards the Response.

The Length field has a size of two bytes and represents the length, in bytes, of the whole EAP packet to which it pertains, including both header and payload.

The Type field has a size of one byte and identifies the type of authentication mechanism that is being used or to be used during the EAP conversation. The EAP Response Type field might be different from the Request, in which case indicates that a Type is unacceptable to the supplicant, sending a Nak (Not Acknowledged). The initial definition included methods such as MD5-Challenge, One Time Password and Generic Token Card, but currently there are many more¹.

The Payload field has a size of zero or more bytes and its content varies with the authentication mechanism, or more specifically, the Type field. This field does not exist if the Code field is Success or Failure.

2.1.2 EAP: Methods

As said, EAP is a meta protocol that encapsulates authentication mechanisms, or EAP methods. The most commonly used methods include EAP-TLS [6], EAP-TTLS [7], EAP-PEAP [8], EAP-LEAP², EAP-SIM [9] and EAP-AKA [10]. The methods differ in aspects such as usability, efficiency, flexibility, and many others.

Table 2.1 presents the cons and pros of such EAP methods.

¹<https://www.ietf.org/assignments/eap-numbers/eap-numbers.xml#eap-numbers-4>

²http://www.cisco.com/c/en/us/products/collateral/wireless/aironet-1200-series/prod_qas0900aecd801764f1.html

Table 2.1: EAP Method Comparison

EAP-Type	Advantages	Disadvantages
EAP-TLS	Mutual authentication through certificates; Blocking user via certificate revocation;	Lack of user identity protection; User certificate configuration is necessary
EAP-TTLS	Mutual authentication; Allows the client to authenticate himself with any authentication method (not only EAP);	
EAP-PEAP	Mutual authentication; Client identity protection on phase 2; Allows the client to authenticate himself using any EAP, in a secure channel;	
EAP-AKA	Mutual authentication; Client authenticates with USIM; Secure PPP authentication model;	RADIUS needs access to HLR;
EAP-SIM	Mutual authentication; Client authenticates with SIM from GSM;	Network impersonation (if some GSM triplets are known); Some A3/A8 algorithms have vulnerabilities;

2.1.3 EAP: Generic EAP Conversation

As shown in Figure 2.2, the conversation starts when the authenticator issues an Identity Request as the initial Request to the supplicant, querying the identity of the latter, to which it sends an Identity Response, containing an identity, used to uniquely identify the supplicant. After that, the authenticator sends an EAP Request containing an EAP Method Type and the supplicant can reply in two ways. It can reply with an EAP

Response of the same type, in which case the pair Request-Response are transmitted as many times as needed to conclude the authentication mechanism (the number of exchanges depends on the EAP method being used). Otherwise, it can reply with a Nak, indicating that the Type solicited by the authenticator is unacceptable to the supplicant and through which it also requests a Type that the supplicant itself accepts, and the authentication proceeds with consequent exchanging of Request-Response pairs. Either way, an EAP Success or EAP Failure message is transmitted at the end, indicating a successful or failed authentication.

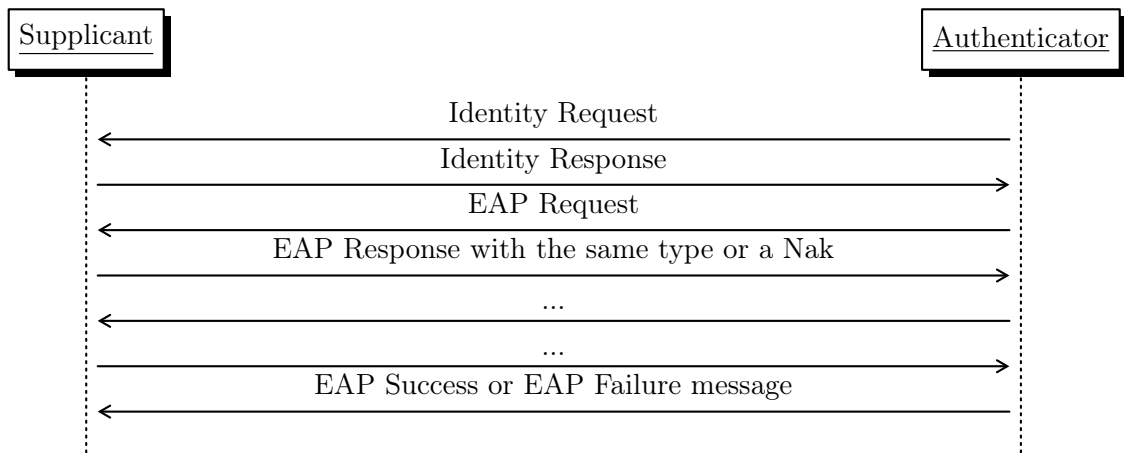


Figure 2.2: Example of an EAP Conversation

2.2 IEEE 802.1X

IEEE 802.1X is an IEEE standard used for port-based network access control [4]. It provides a mechanism for devices to authenticate themselves in LANs or WLANs, based on EAP. This standard allows mutual authentication to be performed for both end-user station (supplicant) and the network, as well as the generation and distribution of session keys, and it was originally conceived for corporate cabled networks, and later extended for wireless networks.

2.2.1 802.1X: Entities

The 802.1X authentication generally involves three intervening parties (see Figure 2.3):

- Supplicant: equipment that intends to attach to the LAN/WLAN. It can be a piece of software that provides credentials to the authenticator;
- Authenticator: network device (*e.g.* an Access Point) that controls the state of the supplicant's access port to the network. It acts as a safety guard to the protected network;
- Authentication Server: central server that leads the mutual authentication process between supplicants and the network (*e.g.* RADIUS).

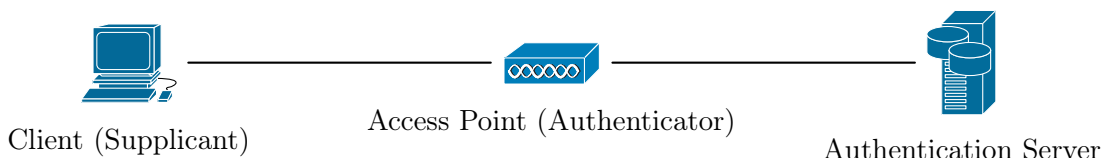


Figure 2.3: 802.1X Basic Architecture

2.2.2 802.1X: Controlled and Uncontrolled Ports

The 802.1X terminology defines that for every authentication session there are two logical ports on the authenticator that control the access to the network. The uncontrolled port does not impose any restriction in terms of data exchange through it, but only allows traffic to be sent and received to perform authentication and key distribution, to and from the authentication server. The controlled port has distinct states and depending on what state it is currently on, it allows or not traffic to go through it:

- Authorized state: allows network traffic to ingress to or egress from the controlled port;

- Unauthorized state: prevents network traffic to ingress to or egress from the controlled port;

The controlled port is initially on "unauthorized" state and upon a successful 802.1X authentication process, it changes its state to "authorized", allowing the end-user station to communicate with the network normally.

2.2.3 802.1X: Authentication Process

In WLANs, when a supplicant associates with an access point, it is bound to two 802.1X logical ports, uncontrolled and controlled port. After the 802.1X authentication, the communication between the supplicant and the network behind the access point is done through the controlled port. This type of authentication is done along three main stages (see Figure 2.4):

1. 802.11 Discovery and Association: On this stage, the supplicant connects to the access point, with the normal 802.11 process of network discovery, OSA authentication and association between the supplicant and the access point. At the end, the controlled port remains in the "unauthorized" state;
2. EAP Authentication/Dialogue: On this stage, the supplicant sends an EAPoW Start (for wireless networks) and EAP messages are exchanged, including the Identity exchange as well as pairs EAP Request-Response and RADIUS Request-Challenge, until the EAP method is completed, with an EAP Success or Failure Message. Generally, on this stage occurs mutual authentication and session key distribution between the supplicant and the authentication server. The authenticator relays the dialog between these two entities, which is the only traffic allowed on the uncontrolled port. In the end, the controlled port remains in the "unauthorized" state.

The EAP messages must be encapsulated to be allowed to be transmitted between parties. In 802.1X, as we can see in Figure 2.4, between the supplicant and the authenticator, EAP messages are encapsulated in EAPoL frames (EAP over LAN), or more precisely, EAPoW frames (EAP over Wireless). Between the authenticator and the authentication server, EAP messages are encapsulated within an application protocol (*e.g.* RADIUS).

3. Four-Way Handshake: On this stage occurs mutual authentication and session key distribution between the supplicant and the authenticator. It involves exchanging four messages and in the end, if every step was successful, the controlled port changes to the "authorized" state. This phase is important because the distribution

of session keys guarantees the confidentiality and integrity of the end-user data in the supplicant-AP wireless link.

As soon as the controlled port is on the "authorized" state, the end-user is granted access to the network by the authenticator.

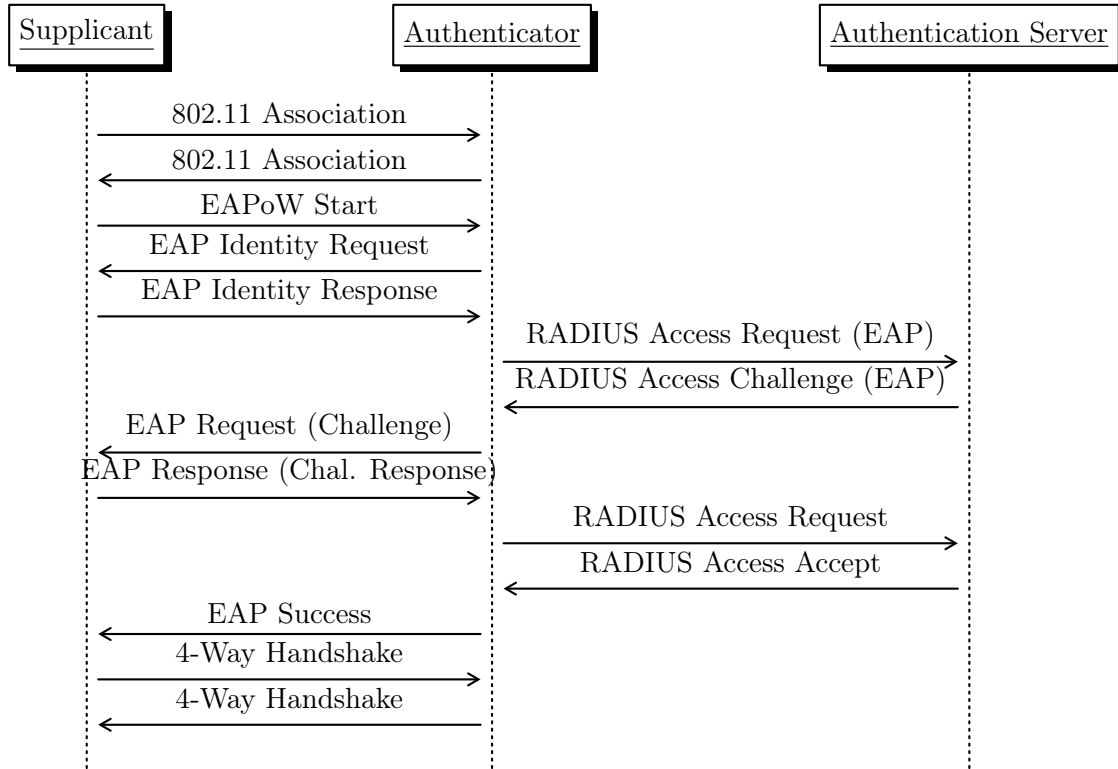


Figure 2.4: Example of an 802.1X Authentication Process

2.2.4 How Linux uses 802.1X

As explained, a supplicant is the entity that seeks to be authenticated in order to connect to a particular network. In Linux the piece of software that plays this role is *wpa_supplicant*. This piece of software is a cross-platform (Linux, Mac OS X and Windows) supplicant with support for WEP, WPA and WPA2 and was designed to be a daemon program, *i.e.*, program that runs in background. It implements key negotiation with a WPA authenticator and it controls the roaming and IEEE 802.11 authentication/association of the wireless driver³. Besides supporting WPA features, it

³https://wiki.archlinux.org/index.php/WPA_supplicant

supports almost every known EAP method. *Wpa_supplicant* also includes a text-based frontend (*wpa_cli*) and a GUI (*wpa_gui*)⁴.

⁴http://w1.fi/wpa_supplicant/

2.3 CAPTIVE PORTALS

A hotspot is a physical location that offers Internet access over a WLAN through the use of a router connected to an Internet Service Provider (ISP). Due to the lack of applicable MAC-layer security solutions, current 802.11 hotspots have one of two security strategies. The first strategy is to use no security whatsoever. This is common for small establishments. The second strategy is to use a captive portal.

A captive portal is a special web page shown before using the Internet normally and it is generally used to present a login page. This is done by a router or a gateway host that does not allow traffic to pass before the user authenticates or performs a specific action.

There are at least three ways of using a captive portal. The first limits access to a set of known users defined generally by a username and a password. The second requires payment before allowing the traffic to pass. The third simply displays the terms of use, to which the user has to comply with, before being granted access.

Generally, a captive portal's operation [11] is defined by the following steps:

1. The supplicant receives an IP address from a DHCP server via the wireless link (as open networks normally do);
2. Traffic is blocked, except the traffic to and from the captive portal server;
3. Any web traffic is redirected to the captive portal server;
4. A web page displaying terms of use, billing information, login screen, or a combination of these is returned;
5. After performing the adequate action, the traffic is unblocked and the user is granted access to the network.

2.3.1 Implementations

This type of paradigm implies the existence of an engine capable of redirecting all traffic to the captive portal server. This piece of architecture may reside on an access point or on a dedicated server. Additionally, the captive portal engine might communicate with a back-end server which stores the users and associated information (*e.g.* LDAP, RADIUS). There are many ways of implementing this paradigm.

ICMP Redirect

ICMP redirects are messages sent to the hosts that notify about an alternative route and so updates their routing information (to follow the route they announce). This way, a supplicant's traffic can be redirected as long as it is not authenticated.

HTTP Redirect

An HTTP Redirect is a message sent by a web server, generally in response to an HTTP GET, instructing the web browser to request for another URL, instead of the originally requested. In this case, the web browser receives as response an HTTP Redirect from the captive portal engine that will always redirect it to the captive portal's web server.

Redirection by DNS (DNS Spoofing)

When a supplicant requests a website, the captive portal engine will make sure that only the DNS server provided by DHCP can be queried by the unauthenticated client or alternatively forward all the DNS requests to a determined DNS server. This DNS server will always return the IP address of the captive portal page as a result of any DNS lookup.

2.3.2 Security Limitations

Depending on the implementations, a captive portal engine can have many limitations in terms of security. Many portals have confidentiality problems, some do not encrypt usernames and passwords whilst others only encrypt these (through TLS tunnelling) during the authentication phase, and thus transmit all succeeding data in clear text. This is done since hotspot operators leave to users the protection of their own communications and many times without informing them.

Another limitation is related with the maintenance of the authenticated session. After an authentication, some implementations allow traffic to pass through the gateway based on IP and MAC addresses. Once an IP and MAC address are found to be authenticated, any listener can spoof these addresses and be allowed a route through the gateway.

2.3.3 Captive Portal Engines

There are many control systems for implementing a portal-based network access control, either open source or paid, each one with its advantages and drawbacks. Some of them are:

1. CoovaChilli: Simple open-source software access controller based on the deprecated ChilliSpot project. It uses RADIUS for access provisioning and accounting⁵;
2. WifiDog: Open-source software access controller that allows users to create a working account directly through the captive portal, granting them 15 minutes to confirm an email. It also records statistics regarding the network and users using it. It allows bandwidth limiting per class, per router and port blocking. It also allows the enforcement of policies based on time of day⁶;
3. Untangle: Open-source software access controller that is very flexible in terms of configuration. Besides having a good GUI for configuration, it allows the captive portal to be displayed only to a subset of the network, or by the user operating system, or by device type, or even define an interval of time on which the portal is required. It also allows the specification of rules to capture or pass traffic and a very easy way to customize the captive portal page⁷;
4. WhizzWifi: Payed software access controller, easy to install and to use, that besides offering the possibility of configuring almost every aspect of a captive portal management system, records analytics, which is very useful for enterprise clients⁸;
5. Facepoint: Payed software access controller that not only offers the most varied analytics (including real time user statistics) but also the possibility of logging in with social network accounts through the captive portal page⁹;

These are some solutions but there are other that use different access control techniques and have different features. On Section 2.3.4, we describe with more detail how one of these systems work, ChilliSpot. ChilliSpot is an open source captive portal engine, used for authenticating users of a wireless LAN and it supports web-based login, which is a *de facto* standard for public hotspot authentication. The authentication, authorization and accounting is handled by a RADIUS server.

2.3.4 ChilliSpot

In this section we describe an experiment done in order to illustrate, for real communications, the behaviour of ChilliSpot.

⁵<https://coova.github.io/CoovaChilli/>

⁶<http://dev.wifidog.org/wiki/Features>

⁷<https://www.untangle.com/shop/captive-portal/>

⁸<http://whizzwifi.com/#features>

⁹<http://www.facepoint.me/wifi-hotspot.html>

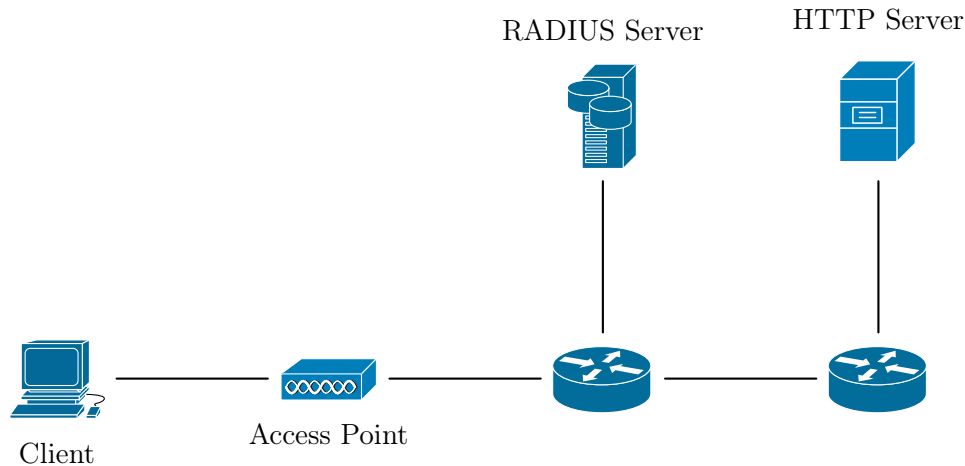


Figure 2.5: ChilliSpot Architecture

For this experiment we used a common terminal, an access point, a RADIUS server and an HTTP server. These two servers were installed in the same physical machine.

The access point had a Linux-based firmware installed, DD-WRT, which includes, among its many features, a ChilliSpot engine, along with other captive portal engines, such as Sputnik¹⁰ and WifiDog¹¹. The AP serves as a mediator for the Internet access, generating the challenges for authentication, serving as a redirector and controlling the authentication session for each terminal that intends to have access to the Internet.

The RADIUS (Remote Authentication Dial-In User Service) server is a piece of equipment that provides centralized Authentication, Authorization and Accounting (AAA). In this case, it only provides authentication, storing in its database the users that have access to the Internet. For this, it was installed a popular open-source RADIUS server, FreeRADIUS and also an open-source database management system, MySQL, to store user information.

The HTTP server is a server that provides the captive portal page. For implementing this server, it was installed an Apache server along with an HTTPS configuration to ensure that every message exchanged between the terminal and this server was over TLS (to protect usernames and passwords, as well as responses to challenges).

¹⁰<http://www.sputnik.com/>

¹¹<http://dev.wifidog.org/>

Authentication Process

In order to know how ChilliSpot works, we performed an authentication while capturing all the traffic in order to analyse the exchanged packets, as shown on Figure 2.6.

First the supplicant sends an HTTP GET, through the browser, to an URL different from the captive portal's (with the intention of accessing some Internet services). The access point answers the supplicant with an HTTP response with a status code 302, which is a temporary redirection to another location. That location is an URL to the captive portal login page. Included on this URL is a 16-byte challenge, generated randomly by the ChilliSpot engine installed on the AP.

After this step, the supplicant's browser sends an HTTP GET to the provided captive portal login URL, along with the challenge, and the HTTP server replies back an OK message with the login page.

After introducing the credentials, the supplicant transmits an HTTP POST that includes the username and the password, as well as the same previous challenge. The server sends back a "Login Result URL" in an OK message which contains a response to the challenge as well as the username. This response is made with the following operations:

$$md5(password, challenge_enc_with_uamsecret)$$

The *password* is the user's one. The access point and the HTTP server share a secret that only they know, *uamsecret*, and the challenge is encoded with this key. *Uamsecret* is a parameter used to encrypt the CHAP-Challenge and the CHAP-Response when it is transferred from the HTTP server to ChilliSpot. This secret is used to prevent dictionary attacks. Also, RADIUS server and Chillispot share a secret key. This key is used to transmit the Challenge and Challenge Response.

The OK message sent by the HTTP server contains HTML meta elements to instruct the web browser to automatically refresh the current page after a given time interval, to a given URL (Login Result URL), creating a timed client-side redirection. This is known as meta refresh¹². As a consequence, the supplicant sends an HTTP GET to the "Login Result URL" (which is sent the ChilliSpot engine to process, on the AP) and the RADIUS authentication process begins. The ChilliSpot engine sends an Access

¹²https://en.wikipedia.org/wiki/Meta_refresh

Request to the RADIUS server, which includes the username and the CHAP-Password, the response to the previous sent challenge. Based on this information, the RADIUS server sends an Access Reject, if the response is not according to what is expected, or an Access Accept, on the other case.

In case of an Access Reject is sent, the access point redirects the supplicant to a failed login page, along with a completely new random challenge, so the process might repeat again.

In case of an Access Accept, the access point redirects the supplicant to a successful login page and then to the original requested page.

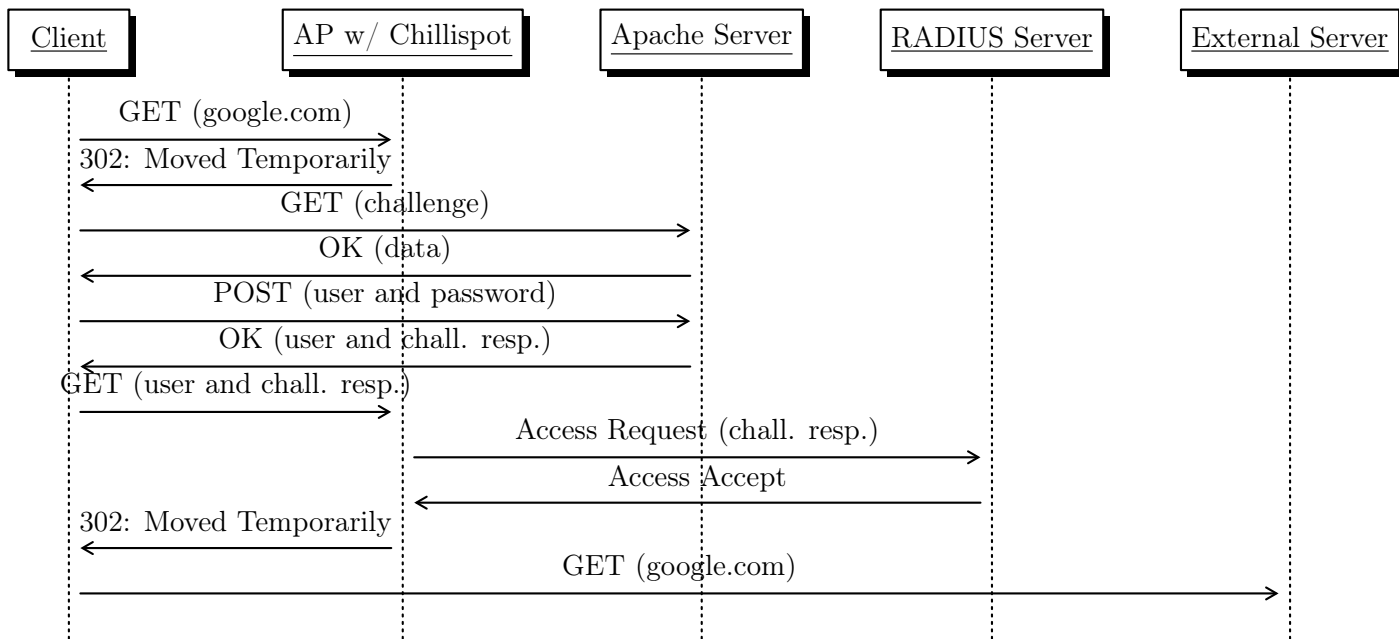


Figure 2.6: ChilliSpot Authentication Process

Three

Related Work

This chapter intends to describe some similar solutions related to the exposed problem. 'We can not solve our problems with the same level of thinking that created them', Albert Einstein

3.1 SECURE MAC-LAYER PROTOCOL FOR CAPTIVE PORTALS

Wi-Fi hotspots are open for public access, and for that reason these environments are not conducive for sharing login credentials or secret key information between the client and the access point. On this solution, the key establishment between client and AP is made using hierarchical identity-based cryptography [12].

Identity-based cryptography is a type of public key cryptography in which an entity's public key is derivable from a known aspect of its entity (*e.g.* an email address). The corresponding private key is generated from a third party, called the Private Key Generator (PKG), based on the entity's identity. However, this requires the setup of a secure channel between the entity and the PKG through which it can send the private key, being the main disadvantage of this type of cryptography. Also, having a single PKG on large network can become a bottleneck, and to solve this scalability problem comes the Hierarchical Identity-based Cryptography, which allows a root PKG to distribute the workload by delegating private key generation and identity authentication to lower-level PKGs.

On this scheme, the MAC address serves as identity, and using hierarchical identity-based cryptography, the PKG generates each user's private key based on the user's MAC addresses and security parameters given by the root PKG.

This solution's protocol [13] is presented for a client device and an access point in a captive portal environment. The manufacturer of each wireless network interface assigns it a unique MAC address as its public key, and the manufacturer uses its PKG

functionality to provide the network interface with a private key corresponding to its MAC address. This protocol uses the public key to exchange the generated private key, which is used to establish a secure tunnel between the AP and the client device. This scheme protects against MAC address spoofing, rogue access points and some MAC-layer Denial-of-Service attacks.

3.2 HOTSPOT 2.0

Hotspot 2.0, also called HS2 or Wi-Fi Certified Passpoint, is a standard for public-access Wi-Fi, developed by the Wi-Fi Alliance and the Wireless Broadband Association. This standard, based on 802.11u, features seamless roaming among wireless networks and also between wireless and cellular networks, increased bandwidth and services-on-demand to end-users. This means that when a subscriber's device is in range of at least one Wi-Fi network, the device automatically connects to it and network discovery, registration, provisioning and access processes are automated¹. This standard is one possible way of circumventing captive portals.

3.3 APPLE'S CAPTIVE PORTAL ASSISTANT

This is a slightly different captive portal implementation, and permits the circumvention of captive portals. Apple's Captive Portal Assistant is a user agent which detects when a captive portal pops up and transparently bypasses the display of the page, against the service's operator, as long as they have access to the correct credentials.

¹<http://whatis.techtarget.com/definition/Hot-Spot-20-HS-20>

Four

Architecture

This chapter intends to give a perspective of the solution and its architecture.

'Those who can imagine anything, can create the impossible', Alan Turing

4.1 OVERVIEW

The authentication on a captive portal associated to a hotspot-based network, as referred before, is generally very insecure, since there is not a standardized secure process to perform it. Indeed, some hotspot service providers might implement their own security on these environments, but to the user there is no guarantee that his transmitted data is being protected.

The solution that we developed intends to take advantage of the 802.1X architecture, and consists on the implementation of an EAP method (for the second phase of 802.1X authentication) complemented with the captive portal paradigm. This EAP method is performed in two stages: first the establishment of a TLS tunnel; then an HTTP-based authentication via a captive portal. The establishment of a TLS tunnel allows the encryption and integrity guarantee of the succeeding data, which that means all the encapsulated HTTP packets are protected. It also assures that the network a user is connecting to is not malicious. The HTTP-based authentication allows the user to input his own credentials, as normally he would when authenticating to a hotspot network with a captive portal.

Usually, each time a user intends to perform authentication, he must open the browser, insert an URL and introduce his credentials on a captive portal. On our solution, upon the success of an HTTP authentication, a certificate is generated, signed by the authentication server and installed on the client. The validity of this certificate is custom, defined by the hotspot service provider. Furthermore, the username is included on the certificate, properly encrypted so that only the server can decrypt it, and use it for accounting purposes, essential on these type of networks. The username encryption

is relevant for protecting the user's identity, since their certificates are exchanged in cleartext. Another advantage of this protocol is that once the certificate is generated, signed and installed on the client, there is no need of contacting the captive portal on any other future session, as long as the certificate remains valid. This mechanism saves time, and is more convenient to a normal user, as well as completely transparent.

AP software modification can be somehow a complex problem that needs to be tackled by captive portals. However, for this protocol (and generally for any kind of EAP protocol), there is no need to modify the AP software, because it already knows how to relay EAP conversations. Then, only the end systems (client and server) need this type of modification, in order for this protocol to work.

Moreover, the captive portal paradigm gives clients the possibility of abusing the network through tunnels over open protocols (*e.g.* DNS). With our protocol, on the contrary, it is impossible to do similar actions since, during the authentication process, the client has no IP address and has necessarily and exclusively to dialogue with the authentication server.

All these aspects and advantages are part of the EAP authentication mechanism presented in the next section, Extensible Authentication Protocol for Secure Hotspots (EAP-SH).

4.2 EXTENSIBLE AUTHENTICATION PROTOCOL FOR SECURE HOTSPOTS

Extensible Authentication Protocol for Secure Hotspots (EAP-SH) is an EAP protocol to be used only in architectures exploring captive portals. Its basic operation consists of two main phases (detailed on Section 4.2.4):

1. First phase, known as the TLS phase, consists on the establishment of a TLS tunnel (through a slightly modified EAP-TLS) between the supplicant and the authentication server. Both sides' authentication works with asymmetric key pairs and public key certificates and, in case the client certificate is invalid or non-existent, the authentication proceeds to a second phase, which runs over a TLS tunnel created with only server-side authentication. In case the client has a valid certificate, the authentication is performed with it;
2. Second phase, known as the Captive Portal phase, consists on the exchange of EAP-SH-formatted messages that can either contain HTTP or certificate data, for

the purpose of presenting to the user the login page and generating a certificate. On this phase, all the messages circulate encrypted over the previously created TLS tunnel. After a successful login, the user obtains a valid certificate, and the process terminates successfully.

The whole process is completely transparent to the user, and it acts as a normal captive portal, needing only to input his credentials and in case his certificate is still usable, having direct access to the hotspot network.

4.2.1 EAP-SH: Message Structure

Both client and server must have a common message format to use when communicating between themselves. On the stages where EAP-TLS messages are exchanged, the message structure is as described on RFC 5216 [6] (Figure 4.1).

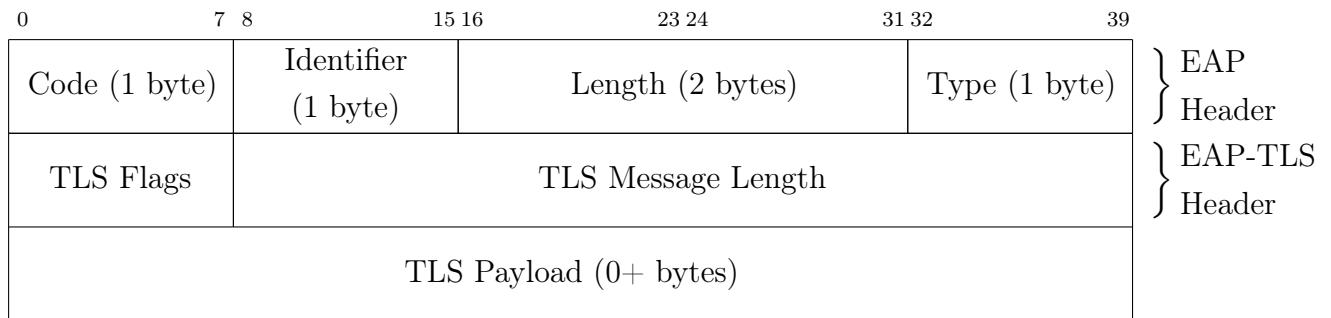


Figure 4.1: EAP-TLS Message Format

EAP-TLS fields are as follows:

- TLS Flags, represented on Figure 4.2, is a 1-byte field, with each bit having its own meaning. The L bit (length included) is set to indicate the four-byte TLS Message length field and is always set on the first fragment of a fragmented TLS message. The M bit (more fragments) is set to indicate if more fragments are to come, and is active on all fragments but the last. The S bit (EAP-TLS Start) is set in an EAP-TLS Start message. The H bit, which is a bit not present in the original EAP-TLS, is set to indicate the start of the second phase. The R bits (reserved) are always set to zero, on this specification. Fragment acknowledges have all of their flag bits set to zero.

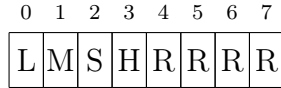


Figure 4.2: Modified EAP-TLS Flags byte for the first stage of the protocol

- TLS Message Length is a 4-byte field and is present on the first fragment of a fragmented TLS message, which means when the L bit is set. It provides the total length, in number of bytes, of the full EAP-SH message that is being fragmented.

On stage 2 (Captive Portal stage), where HTTP and certificate information is exchanged between supplicant and server, messages have a different format, though based on EAP-TLS format. The EAP-SH message structure is shown on Figure 4.3.

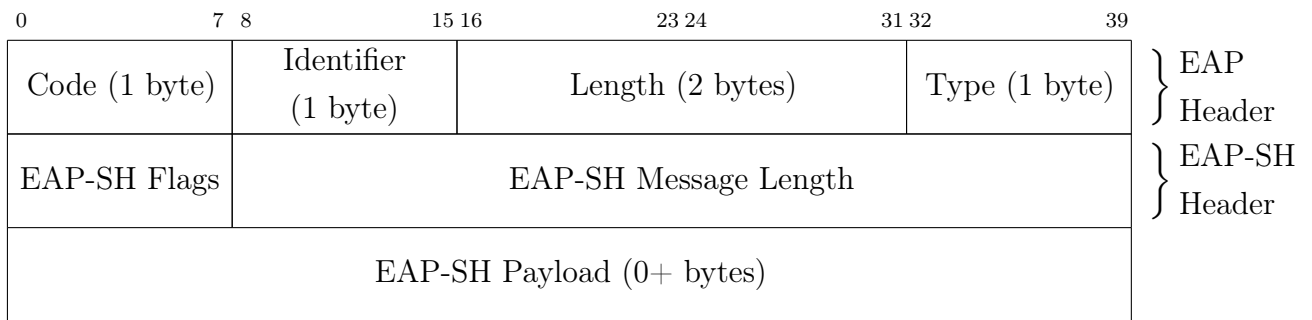


Figure 4.3: EAP-SH Message Format

EAP-SH fields are as follows:

- EAP-SH Flags, represented on Figure 4.4, is a 1-byte field, with each bit having its own meaning (similar to EAP-TLS). The L bit (length included) is set to indicate the four-byte TLS Message length field and is always set on the first fragment of a fragmented EAP-SH message. The M bit (more fragments) is set to indicate if more fragments are to come, and is active on all fragments but the last. The G bit is set to indicate the content of the payload includes an HTTP request (either GET or POST) coming from the client, and so only client EAP messages can have this flag on. For HTTP responses, this bit is set to zero. The C bit is set when the client sends an EAP message with its certificate signing request. The R bits (reserved) are always set to zero by default. Fragment acknowledges have all of their flag bits set to zero.

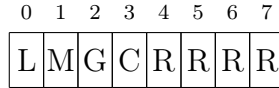


Figure 4.4: EAP-SH Flags byte for the second stage of the protocol

- EAP-SH Message Length is a 4-byte field and is present on the first fragment of a fragmented EAP-SH message, which means when the L bit is set. It provides the total length, in number of bytes, of the full EAP-SH message that is being fragmented.
- EAP-SH Payload is a field that can either contain HTTP information (Request or Response) or a client's certificate. This topic will be explained further on.

4.2.2 EAP-SH: Flowchart

The authentication process begins with the first stage. First, there is a client certificate transmission to test if the certificate exists and it is valid. Then, EAP-TLS starts and the authentication server sends its certificate to the client, who verifies its validity (if the certificate has expired). If the server certificate fails the validation, EAP-TLS terminates immediately, and the authentication process fails. The entity managing the hotspot service has the responsibility to maintain the validity of its servers' certificates. In case the server certificate passes the validation, one out of two situations can happen. In case the initial client validation is successful, the EAP-TLS proceeds with client authentication, and terminates successfully. In case it is not, EAP-TLS proceeds only with server authentication and a second phase begins to obtain a valid client certificate. In either cases, a TLS tunnel is always established.

The second stage serves as a mean of obtaining a valid certificate through the use of easier-to-manage credentials, such as usernames and passwords. This process occurs through the previously established TLS tunnel. For this, the client asks for the captive portal's web page and the server answers the page back to him. As soon as the client introduces the credentials on the web page and sends a POST with the credentials, the server sends back one of two possible pages/results. It can send a failure page, on which it is possible to introduce the credentials and send them again or, in case the credentials are successfully validated, the server sends a success page back, and in response receives a certificate signing request from the client. This certificate is then signed by the server, which is also a Certification Authority, and then sent back to the client. As the created certificate is valid, the authentication process ends successfully. A flowchart representing how this authentication process works is shown on Figure 4.5.

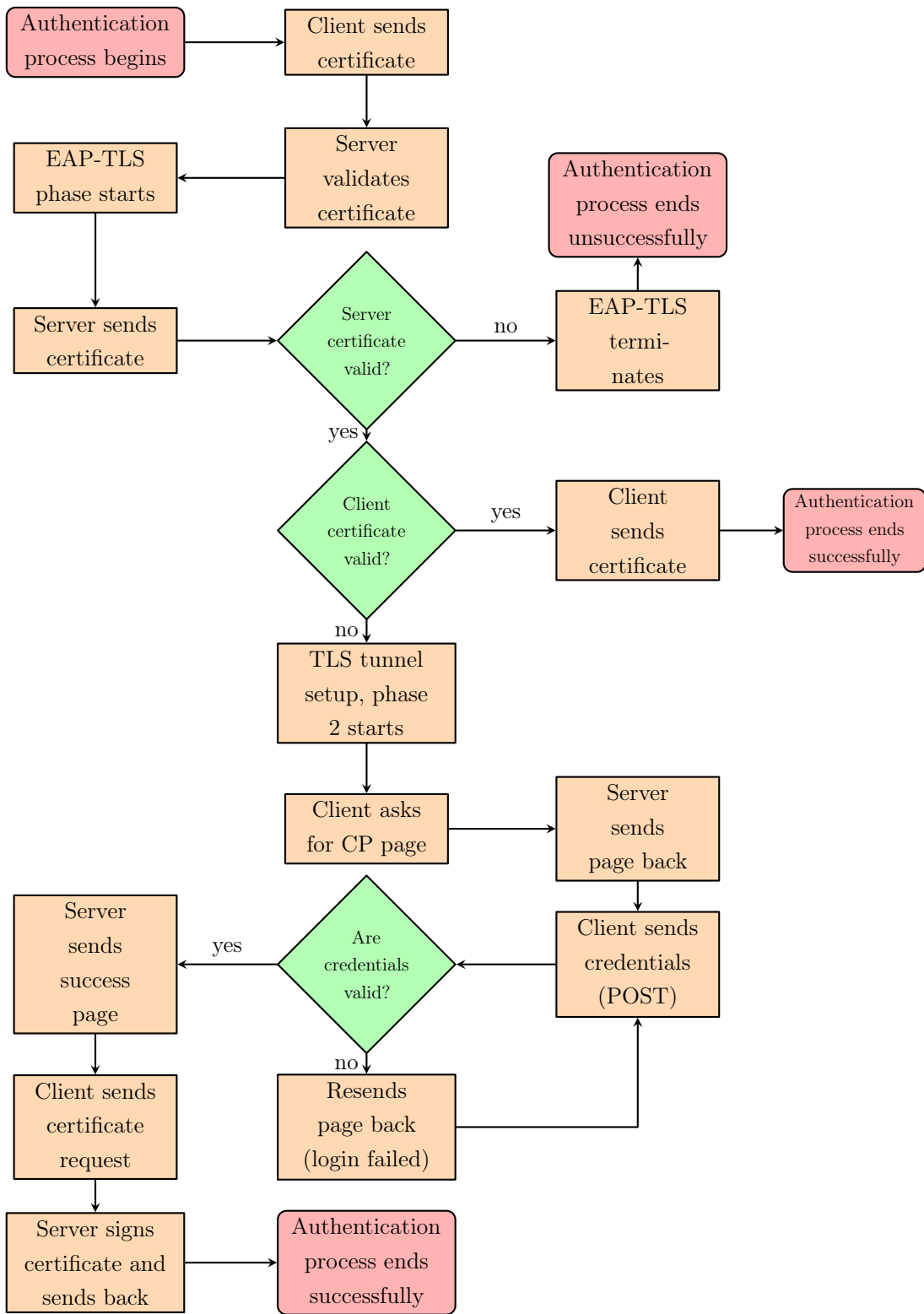


Figure 4.5: EAP-SH Flowchart

4.2.3 EAP-SH: Certificate Infrastructure

Digital certification consists on the emission of public key certificates, which are documents with a predefined structure that contain a public key of a certain entity and a digital signature from the issuing Certification Authority (CA). They possess an expiration date and it can be controlled through a period of validity field indicated on the certificate itself [4].

In the EAP-SH architecture, on the server side we have the server's certificate, signed by the Server Certification Authority, which in turn is signed by the Certification Authority of the entity responsible for this service, the hotspot service provider, signed by a Root Certification Authority, on which is self-signed and trusted. This succession of entities makes a trustworthy certification chain. This means that for the validation of the server's certificate, issued by the Server's CA, which in turn is issued by the hotspot service provider, the public key certificate pertaining to this Server's CA needs to be obtained and to validate it, the public key certificate of the hotspot service provider CA needs to be obtained, and so on, until the Root is reached, which is trusted.

A server CA belongs to a central authentication server. A central authentication server, pertains generally to a determined ISP, and it provides authentication to a service given by this latter (*e.g.* Hotspot Wi-Fi). Each server should have a valid certificate for EAP-SH to work. This means the authentication server is trusted and should be used to login through the provided captive portal. It also means the HTTP server from which the server obtains and transmits the captive portal is trusted. This means that the server plays two roles:

- As a Certification Authority, signing incoming CSR's (explained further ahead) from clients with valid credentials;
- As a server who intends to authenticate himself towards the supplicant, having a certificate for that purpose.

A client user wanting to authenticate himself also wishes to obtain a valid certificate, through Certificate Signing Request (CSR)¹. CSR is an encoded message that is given to a Certification Authority when applying for a public key certificate. It is usually generated on the machine where the certificate will be installed and contains several information that can be included when creating it, such as city, country, email address and name (which can also be used by the provider for statistical purposes). It also contains a public key that will be included in the certificate. In general, it might be

¹https://en.wikipedia.org/wiki/Certificate_signing_request

considered an 'unsigned' version of a certificate. If the supplicant does not have a certificate or if it is not valid (as explained on Section 4.2.2), after its credentials being validated, it sends a CSR to the authentication server. The authentication server CA signs it, and sends it back to the client, and hence the client becomes owner of a valid certificate, issued by the server CA.

This issued client certificate is valid during a variable number of days, defined in the certification policy of the server CA. During this period of time, the client is transparently validated by any hotspot related with the same ISP service, and does not need to introduce his credentials. After this time has passed, it needs to be authenticated again through the introduction of captive portal's credentials.

Upon creating a CSR, the username (client's identity) is inserted in a certificate field (known as Subject Alternative Name) encrypted with the server's public key. The client's identity goes along with the certificate when phase one of EAP-SH is occurring and only the authentication server can decrypt it. This is used mainly for accounting purposes, so the server knows to which client it is talking to. The Subject field can be left empty as long as we use the Subject Alternative Name.

As for the server certificate validity, various circumstances may cause this certificate to become invalid prior to the expiration date. On this case, the certificate is revoked. Normally, a Certificate Revocation List (CRL) is used to check if a certificate was revoked. A CRL is a list of serial numbers of certificates that have been revoked by the issuing CA before their expiration date. In this context, instead of requesting CRLs, which can be a waste of bandwidth, Online Certificate Status Protocol (OSCP) is used. OSCP [14] is a protocol used for obtaining the revocation status of a certificate, which comes on an OSCP Response, a reply for a Request made to OSCP servers. However, the client does not have any access to the network to get this response in order to check if the server certificate revoked. To solve this problem, a TLS extension is used, the TLS Certificate Status Request [15], also known as OSCP stapling. Such extension allows the server to append an OSCP Response to a message of the TLS handshake, and therefore the client does not need to contact the CA of the server certificate to get it.

Figure 4.6 shows an example of the certificate hierarchy model of the solution. On this figure, we have two Portuguese ISPs, MEO and NOS. Their certificates are signed by a Root CA, which might not be necessarily the same, contrary to what the figure suggests. Each of these ISPs provides hotspot services: MEO service is MEO Wi-Fi, and NOS service is FON_ZON_FREE_INTERNET. The credentials used to

authenticate in a hotspot network, say MEO Wi-Fi, generate a certificate that can be used to easily authenticate in any MEO Wi-Fi hotspot. As said before and in order to give an example, FON_ZON_FREE_INTERNET CA certificate and the authentication server's certificate are located in the same central server, who acts as a CA and as a server that needs to authenticate himself.

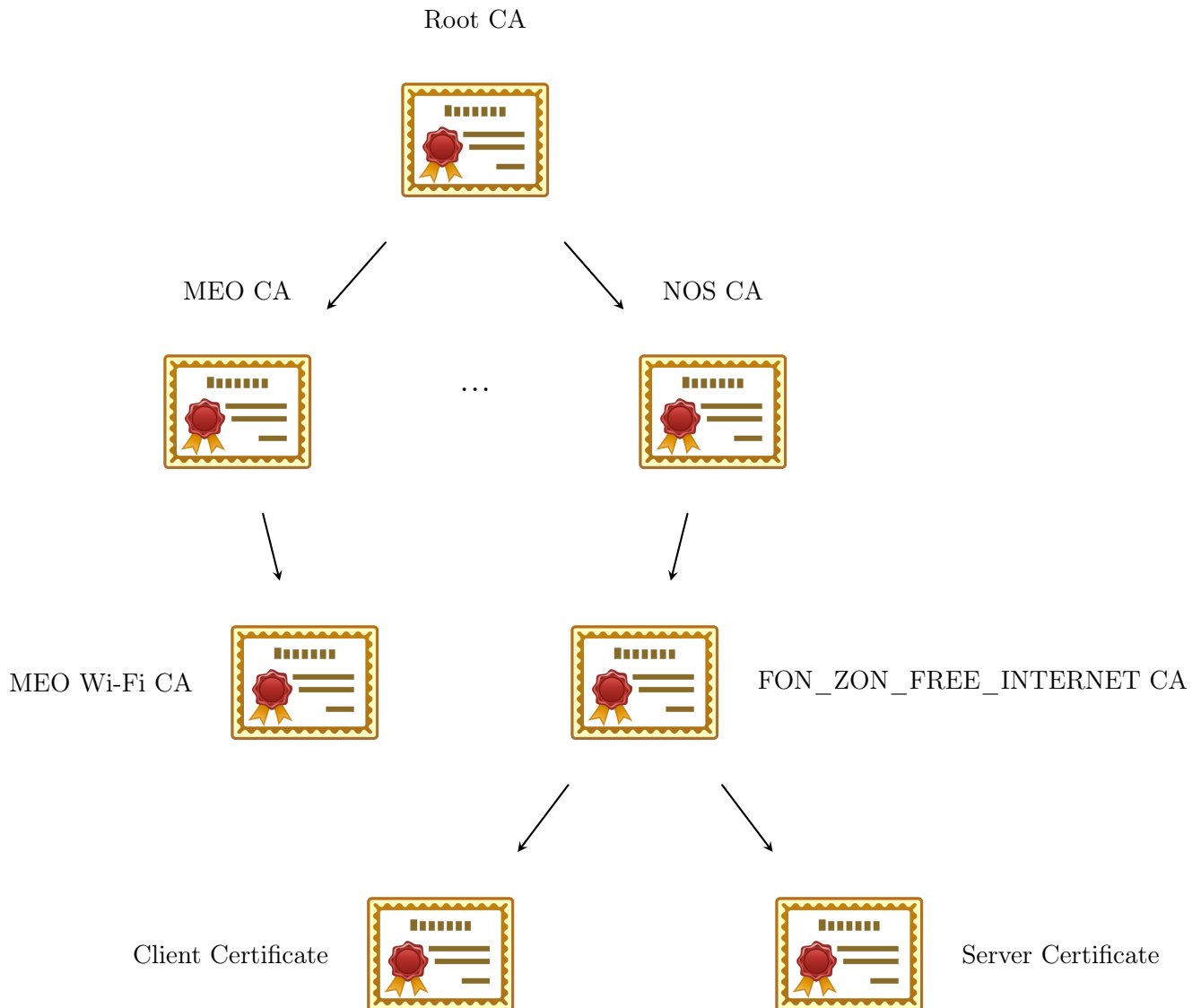


Figure 4.6: EAP-SH Certification Hierarchy Model. The server certificate is used by an 802.1X authentication server to authenticate itself. Note, however, that the same server uses the credentials of its issuing CA for issuing the clients' certificates and its own certificate.

4.2.4 EAP-SH: Detailed Operation

Phase One: EAP-TLS

As said before, EAP-SH starts with normal EAP-TLS operation. Stage 1 of EAP-SH consists on the mutual authentication via TLS and if successful, EAP authentication terminates. Figure 4.7 illustrates how EAP-TLS works when successful (note that the client certificate request is set).

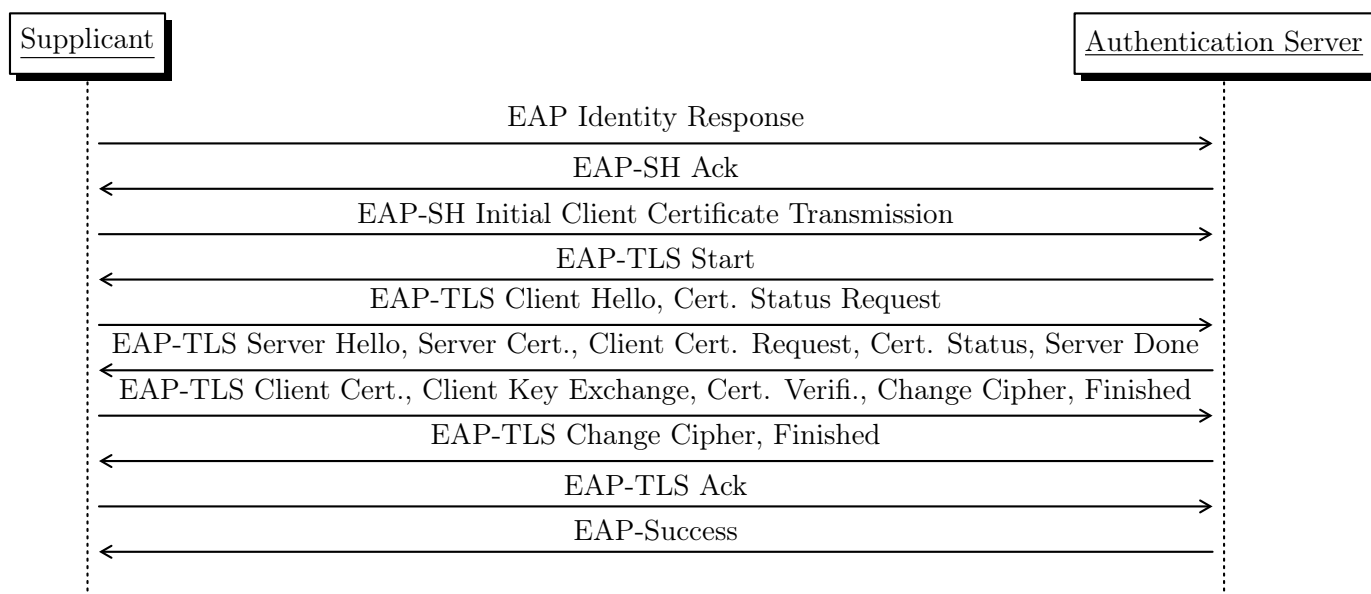


Figure 4.7: EAP-SH Phase 1 Message Exchange - Success

Assuming an 802.1X architecture, after the 802.11 association with the authenticator (Access Point), the client was not yet given any IP address, and so does not have any access to the network. Then EAP authentication phase of 802.1X begins on the first stage of EAP-SH:

1. The first message exchanged is EAP Identity Response, replying to EAP Identity Request sent by the authenticator (though not shown on Figure 4.7);
2. The supplicant then sends its certificate on the Initial Client Certificate Transmission. The server uses this message to validate the client's certificate and here it decides whether or not to send the Client Certificate Request. On this case, the client's certificate is valid;

3. EAP-TLS Start (S bit set on TLS Flags field) is sent to indicate the start of the EAP-TLS handshake (and for that has no data);
4. The Client Hello is a message containing a list of cipher suites (combination of cryptographic methods) and compression methods that the clients supports. In addition to this message, comes the Client Status Request, which is a request, sent to the server, for the use of OSCP stapling;
5. The next message contains the Server Hello, the Server Certificate (which has the name and public key of the server) and the Client Certificate Request. This request is optional on EAP-TLS, but on this context it is mandatory. There is also a Certificate Status message containing OSCP information queried by the authentication server to an OSCP server and a Server Hello Done. At this point the server waits for the client's next step;
6. The client replies with his certificate and sends the Client Key Exchange. This last message is a pre-master secret sent encrypted to the server (with its public key) to compute a master secret. Also, the client sends a hash of all received and sent messages to the server, signing it with its private key, to prove that it is the legal owner of the identity present in the certificate. Finally, it sends a Change Cipher Spec message (whose goal is to change the connection state, if needed) and a Finished message;
7. And the server sends a Change Cipher Spec and Finished messages.

The process finishes with an EAP-Success message and both parties are authenticated. In case EAP-TLS only performs server authentication, the client needs to be authenticated and one of two possible scenarios can lead to that.

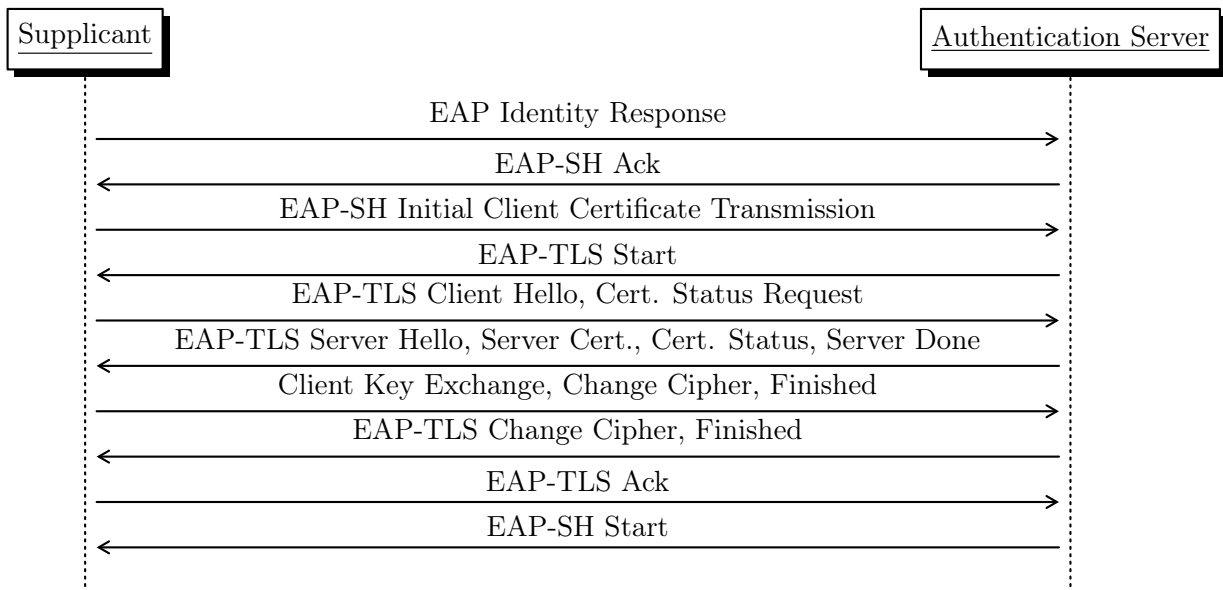


Figure 4.8: EAP-SH Phase 1 Message Exchange - Failure

The first scenario is the scenario where the client certificate does not exist. On the EAP-SH Initial Client Certificate Transmission message, the client sends a "null certificate", a certificate with length equal to zero. The server, upon receiving this, configures itself to not require a client certificate and starts the EAP-TLS process, setting up a TLS tunnel that will be used between the client and the portal, encrypting all data exchanged. Finally, the client receives an EAP-SH Start from the server. After this, both server and client are ready for the Captive Portal phase to start.

The second scenario is the scenario where the client certificate is not valid. On the EAP-SH Initial Client Certificate Transmission message, the client sends its certificate. After this message, the server immediately performs the client certificate validation, and if validated unsuccessfully, the server configures itself to not require a client certificate and starts the EAP-TLS process. Apart from this, everything is the same as in the first scenario.

Phase Two: Captive Portal

This phase features the captive portal's transmission, credential information as well as certificate information (CSR and a signed client certificate). For the captive portal's transmission, the exchange of HTTP messages between the client and the server implies that there must be a TCP connection since HTTP is ran over this protocol. For this reason, there were a couple of mechanisms designed for this to work. As the client machine does not have an IP address, it cannot send and receive HTTP directly. The

solution for this problem was pretty simple: perform encapsulation of HTTP messages into EAP requests and responses. However, there was still a need to establish TCP listening ports and for this:

- A socket on the client is instantiated and bound to a port, listening and accepting TCP connections, objectively from the browser. This socket is instantiated as long as this stage is active. The browser is automatically opened and forced to establish a TCP connection and send a request for the captive portal through this specific socket.
- A socket on the authentication server is instantiated and initiates a TCP connection with the HTTP server (note that the authentication server has an IP address and so the HTTP server can be elsewhere than locally). This socket is used to transmit and receive HTTP requests from both client and web server. When the authentication server receives an HTTP request from the client (encapsulated in EAP) the socket sends it to the HTTP server, receiving a response that can then be fragmented over EAP and sent back to the client.

Figure 4.9 elucidates how this works.

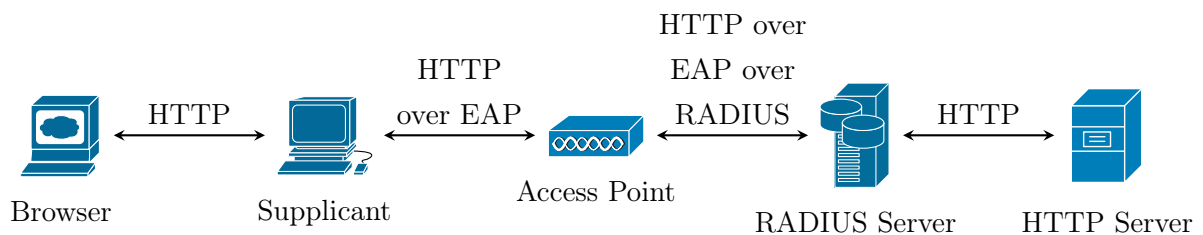


Figure 4.9: EAP-SH Phase 2 - HTTP Protocol Mechanism

The following figures (Figures 4.10 and 4.11) describe in more detail how the message exchange on this phase is processed.

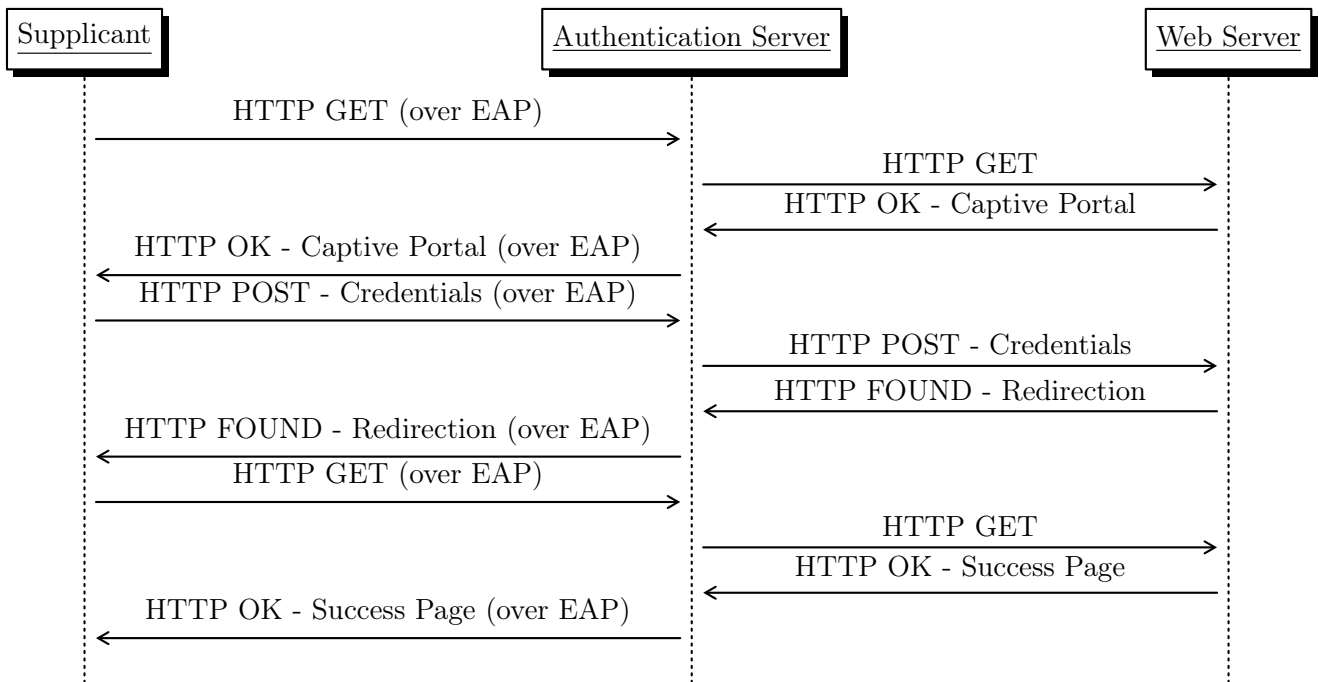


Figure 4.10: EAP-SH Phase 2 Message Exchange - Captive Portal Login Success

The packet exchange starts as soon as the client receives the EAP-SH Start. First the client creates the socket bound to a port (as referred before) and then the client opens a browser automatically, redirecting the HTTP request (GET) to the listening port, establishing a TCP connection. Internally, the supplicant fragments the request if necessary, and sends the fragments encapsulating them in EAP responses, one at a time. On the first fragment of this message, the G bit (Figure 4.4) is set to indicate the message is an HTTP request. Also, for each of these fragments, the corresponding EAP-SH ACK is received. EAP-SH Ack is a message that has no payload and its flags are all set to zero (similarly to EAP-TLS). For the last fragment, the server does not reply with EAP-SH ACK but instead with the reply message he is supposed to (HTTP OK or any message with a different HTTP code).

The server, after receiving all fragments, performs defragmenting on the request message and sends it to the HTTP server through the created server socket, receiving an HTTP response back with the captive portal contents. The authentication server then fragments it, if necessary, and the process is repeated the same way as it does on the client (note that the G bit is not set, for this is not an HTTP request).

Upon receiving the entire response and defragmenting it, the client replies (through its socket) to the browser with the recently received response. The user introduces its normal credentials onto the captive portal and sends a POST, encapsulated in EAP.

The request reaches the authentication server, who then relays it to the web server. The HTTP server then validates the credentials, consulting its user database, and sends a page according to the result. On one hand, if the credentials are not valid, the web server sends back the captive portal with an "authentication failed" message and the user has the opportunity of submitting credentials again. On the other hand, if the credentials are valid, the server sends back to the client a success page, as shown in Figure 4.10.

Both authentication server and supplicant need to have some signalling concerning the authentication's success by the web server. For that, the last HTTP message comes with an additional HTTP header field, *X-username*, where the username is inserted. This username is also used so the authentication server is aware of the current user being accounted. In case this field is empty, the authentication was a failure.

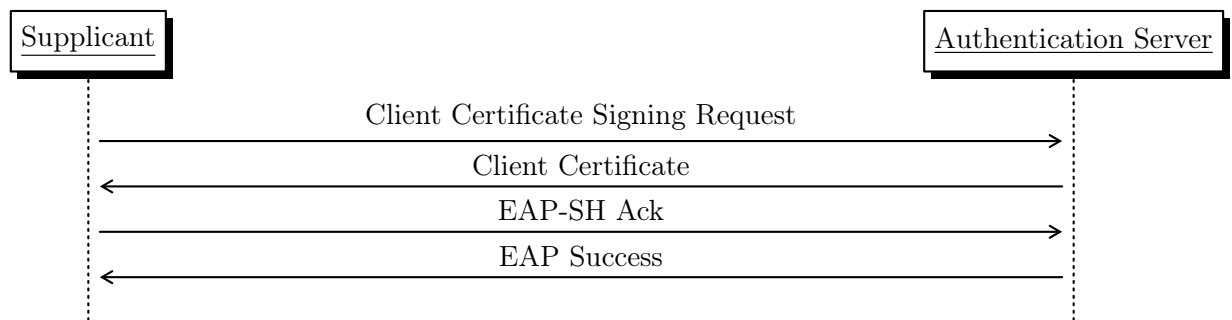


Figure 4.11: EAP-SH Phase 2 Message Exchange - Certificate Signing and Install

Considering a successful validation, a Certificate Signing Request is created by the supplicant and sent to the authentication server, with proper fragmentation. This CSR also contains the supplicant's identity encrypted with the authentication server's public key, inserted on a specific field: Subject Alternative Name. Also, on the first fragment of the message the C bit is set (Section 4.4) to indicate the content of the message is a certificate. The authentication server, upon reception, signs the received certificate with its CA private key and sends it back to the supplicant. From this point on, the certificate is valid, for a number of days according to the certificate policy, and the user can successfully be authenticated by the authentication server without having to contact the captive portal. Finally, the server sends an EAP Success to the authenticator, allowing access to the network as soon as the four-way handshake is complete (to simplify we consider the authenticator and the supplicant the same identity, in Figure 4.11).

Five

Implementation

This chapter intends to give a detailed description about the solution's implementation. I have not failed. I've just found 10,000 ways that won't work', Thomas A. Edison

5.1 WPA SUPPLICANT

As discussed before, the supplicant side on Linux is played by a daemon program that controls the wireless networking called WPA Supplicant¹. WPA Supplicant is a cross-platform supplicant that supports WEP, WPA and WPA2, and it is the IEEE 802.1X component that is used in client stations. For this, it supports a wide variety of EAP methods². The following steps are used when a machine associates to an AP:

1. WPA Supplicant requests the kernel network driver to scan neighbour BSS's (Basic Service Sets), identified by an SSID;
2. WPA Supplicant selects a BSS based on its configuration;
3. WPA Supplicant requests the kernel network driver to associate with the chosen BSS;
4. If WPA-EAP (Enterprise environments): the 802.1X supplicant completes the EAP authentication with the authentication server and a Master Session Key (MSK) is generated;
5. If WPA-PSK (SOHO environments): WPA Supplicant uses Pre-Shared Key (PSK) as the master session key;
6. WPA Supplicant completes the WPA 4-Way Handshake and Group Key Handshake with the AP;
7. Data packets can now be received and sent.

WPA Supplicant is written in C language and in order to integrate our protocol, we downloaded its source code with version 2.6³. Each EAP method is implemented as

¹http://w1.fi/wpa_supplicant/wpa_supplicant-devel.pdf

²https://linux.die.net/man/8/wpa_supplicant

³<https://w1.fi/cgit>

a separate module, usually as one C file named "*eap_<method_name>.c*". All EAP methods use the same interface between the supplicant state machine and method-specific functions, and method registration is done through this interface, as shown on Code Snippet 1.

```
int eap_peer_sh_register(void)
{
    struct eap_method *eap;

    eap = eap_peer_method_alloc(EAP_PEER_METHOD_INTERFACE_VERSION,
                               EAP_VENDOR_IETF, EAP_TYPE_SH, "SH");

    if (eap == NULL)
        return -1;

    eap->init = eap_sh_init;
    eap->deinit = eap_sh_deinit;
    eap->process = eap_sh_process;
    eap->isKeyAvailable = eap_sh_isKeyAvailable;
    eap->getKey = eap_sh_getKey;
    eap->get_status = eap_sh_get_status;
    eap->has_reauth_data = eap_sh_has_reauth_data;
    eap->deinit_for_reauth = eap_sh_deinit_for_reauth;
    eap->init_for_reauth = eap_sh_init_for_reauth;
    eap->getSessionId = eap_sh_get_session_id;

    return eap_peer_method_register(eap);
}
```

Code Snippet 1: EAP-SH method registration

This allows new EAP methods to be added without modifying the core EAP state machine implementation. Additionally, a new EAP type number had to be added to the EAP type definitions. As of today, numbers 56 to 191 are unassigned⁴, and so we chose to attribute to EAP-SH the type number 56. So, for EAP-SH integration, all we needed to do was to create a C file and register its methods, declare its type number and change the Makefiles to include the new C file.

As shown before, our protocol incorporates an EAP-TLS phase, similarly to PEAP or TTLS. For that reason, we adapted the WPA Supplicant's PEAP modules into our protocol, incorporating functionalities to our created modules from PEAP and adding

⁴<http://www.iana.org/assignments/eap-numbers/eap-numbers.xhtml>

new ones according to our defined protocol architecture.

5.1.1 Configuration

WPA Supplicant is configured using a text file (*wpa_supplicant.conf*⁵) that lists all accepted networks and security policies. It can include one or more networks blocks, that include all the necessary configuration for that network. Code Snippet 2 shows configurations for two types of network: home network and enterprise network. Taking a look at the enterprise network configuration, it is possible to configure the EAP method as well as all the certificate information needed for EAP to work.

```
network={
    ssid="home"
    scan_ssid=1
    key_mgmt=WPA-PSK
    psk="passphrase"
}

network={
    ssid="enterprise"
    scan_ssid=1
    key_mgmt=WPA-EAP
    pairwise=CCMP TKIP
    group=CCMP TKIP
    eap=TLS
    identity="user@example.com"
    ca_path="/etc/cert/"
    client_cert="/etc/cert/user.pem"
    private_key="/etc/cert/user.prv"
    private_key_passwd="password"
}
```

Code Snippet 2: Example of two network configurations: home and enterprise network

In our protocol, the network block comprises a number of fields that contain information used when authenticating to a network. Code Snippet 3 shows a configuration example of a network that uses EAP-SH for authentication:

- **ssid**: Sequence of characters that uniquely identifies a WLAN;
- **key_mgmt**: Accepted authenticated key management protocol;
- **eap**: EAP method used when authenticating to this network;

⁵https://w1.fi/cgit/hostap/plain/wpa_supplicant/wpa_supplicant.conf

- **identity**: EAP peer identity, included on Response Identity message (can be random);
- **ca_path**: Path pointing to the directory containing CA certificates files in an OpenSSL format;
- **client_cert (optional)**: Path to the client certificate file, in an OpenSSL format. This file is optional, however it is used on the first phase of EAP-SH. In case it is not present or invalid, a new is generated after performing authentication and added to the file automatically;
- **private_key (optional)**: Path to the client private key file, in an OpenSSL format. It is added automatically along with the client certificate file.

```
network={
    ssid="example-wifi"
    key_mgmt=WPA-EAP
    eap=SH
    identity="bob"
    ca_path="/etc/ssl/certs/"
    client_cert="/home/nuno/ssl/certs/nuno0.pem"
    private_key="/home/nuno/ssl/privkeys/nuno0.pem"
}
```

Code Snippet 3: Network configuration using EAP-SH

5.1.2 Data structures

EAP-SH needs data structures in order to store state variables, necessary temporary information or TLS data structures. The following code snippet shows our data structure and its internal variables.

```

struct eap_sh_data {
    struct eap_ssl_data ssl;
    eap_sh_state_t state;           //new
    int client_socket_fd;         //new
    int client_conn_socket_fd;    //new

    struct wpabuf *resp_out;
    size_t resp_out_pos;
    size_t resp_out_limit;

    struct wpabuf *resp_in;
    size_t resp_in_pos;

    char* certificate;           //new

    char *username;              //new

    int no_certificate;           //new
    int send_certificate;        //new

    u8 *key_data;
    u8 *session_id;
    size_t id_len;
}

```

Code Snippet 4: EAP-SH Data Structure

- **ssl**: Stores TLS connection context data and buffers for TLS messages to be fragmented or to be reassembled;
- **state**: State variable that stores the current state of the EAP-SH process (explained further);
- **client_socket_fd**: Socket file descriptor pertaining to the socket that is listening to connections;
- **client_conn_socket_fd**: Socket file descriptor pertaining to the current connection socket;
- **resp_out**: Buffer that stores the full EAP packet to be transmitted;
- **resp_out_pos**: Position on the buffer still left to be transmitted;
- **resp_out_limit**: Fragment maximum size;
- **resp_in**: Buffer that stores the full EAP packet currently being received;
- **resp_in_pos**: Position on the buffer still left to be received;
- **certificate**: Stores a generated Certificate Signing Request;
- **username**: Username (if the login is completed) to be included in the created certificate, encrypted;

- `no_certificate`: Indicates if there is no certificate, after parsing the configuration file;
- `key_data`: Master secret used to encrypt the transmitted packets;
- `session_id`: Session identifier derived from TLS;
- `id_len`: Session identifier length;

5.1.3 EAP-SH Supplicant States

The supplicant needs to know on what point of the EAP-SH protocol it is so to respond properly to the server. This is done through an internal state machine. The code snippet below represents the list of all possible states.

```
typedef enum {
    EAP_SH_SEND_FIRST_CERT,
    EAP_SH_TLS,
    EAP_SH_START_PHASE2,
    EAP_SH_RECEIVING_CP_PAGE,
    EAP_SH_CP_PAGE_COMPLETE,
    EAP_SH_SENT_POST,
    EAP_SH_RECEIVING_FAILURE_PAGE,
    EAP_SH_RECEIVING_SUCCESS_PAGE,
    EAP_SH_SUCCESS_PAGE_COMPLETE,
    EAP_SH_FAILURE_PAGE_COMPLETE,
    EAP_SH_RECEIVING_CERTIFICATE,
    EAP_SH_CERTIFICATE_COMPLETE,
    EAP_SH_DONE
} eap_sh_state_t;
```

Code Snippet 5: EAP-SH Supplicant States

EAP-SH starts on the `EAP_SH_SEND_FIRST_CERT` state. On this state, the supplicant sends its certificate, and updates its state to `EAP_SH_TLS`, which means that the protocol is now processing the authentication request on the first phase (EAP-TLS phase). As soon as the TLS tunnel is set up, and the certificate is invalid, the captive portal phase starts (`EAP_SH_START_PHASE2`) and the HTTP packets from the portal page are received and acknowledged until all are received (`EAP_SH_CP_PAGE_COMPLETE`). The supplicant sends its credentials through an HTTP POST (`EAP_SH_SENT_POST`) and it can either receive a success page (`EAP_SH_SUCCESS_PAGE_COMPLETE`) or a failure page (`EAP_SH_FAILURE_PAGE_COMPLETE`). On the latter, the client can re-send its credentials, and so re-send an HTTP POST. On the

first, the client generates a CSR, sends it, and expects receiving the certificate from the server (`EAP_SH_RECEIVING_CERTIFICATE`), until is it complete (`EAP_SH_CERTIFICATE_COMPLETE`). Finally, the process terminates and the user is successfully authenticated (`EAP_SH_DONE`).

When generating a CSR, a number of fields are initialized. Such fields, on our implementation, are:

- Country: Client's country of residence;
- Subject Alternative Name: Encrypted client's identity;
- Public key: Certificate public key.

More information on these types of certificates can be stored by the client when generating them (see Section 4.2.3), however only these three are considered on our current implementation.

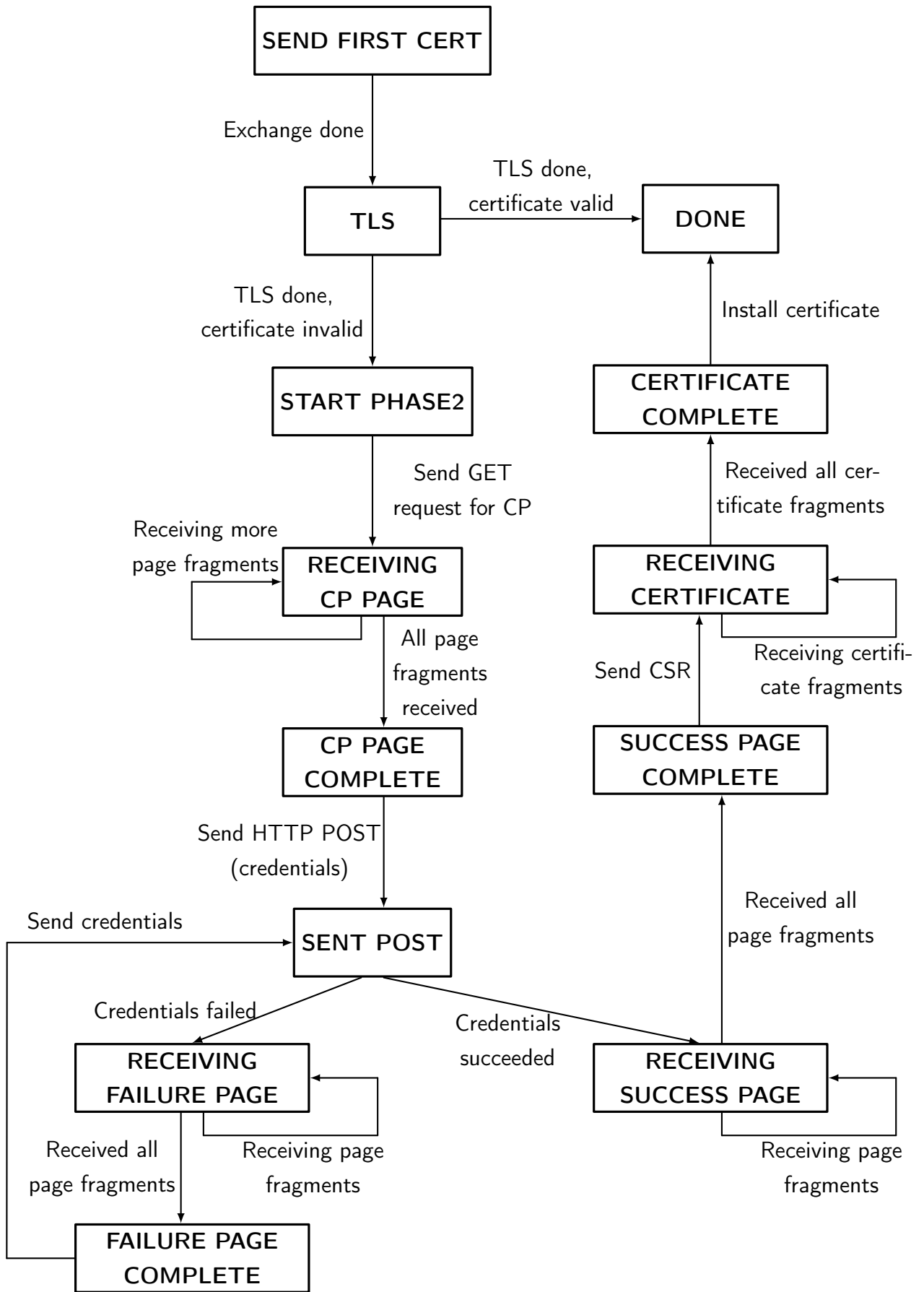


Figure 5.1: EAP-SH: Supplicant State Machine

5.1.4 Fragmentation and Defragmentation

The Internet, as it is today, supports a maximum packet size for transmission, known as the Maximum Transmission Unit (MTU). In most LANs, the MTU has a value of 1500 octets. For this reason, some protocols break and reassemble datagram packets in order to be transmitted in smaller pieces (known as fragments). These processes are known as fragmentation and defragmentation.

On the first phase of EAP-SH, which corresponds to the setup of a TLS tunnel, the fragmentation and defragmentation is based on the flags set on the transmitted packet and so handled exclusively by EAP-TLS module. As referred on Section 4.2.1, there are 2 bits on the Flags field of TLS that control these processes: the L bit (Length Included bit), which generally is set on the first fragment, and the M bit (More Fragments bit), which is set on all but the last fragment. Four scenarios can occur:

- Unique fragment: only the L bit is set;
- First fragment (not unique): both the M and L bits are set;
- Fragment (not first or last): only the M bit is set;
- Last fragment: no flags are set.

On the second phase of EAP-SH, the mechanism is exactly the same. Also, as seen on Figure 5.1, the states are used to make the protocol process a message, when it is fully received, such as a page or a certificate.

5.2 RADIUS

The authentication server is implemented by a Remote Authentication Dial-In User Service (RADIUS) server. RADIUS [16] is a protocol that provides centralized Authentication, Authorization and Accounting management for users that connect and intend to use a network service. FreeRADIUS⁶ is a popular open source RADIUS server and it was the implementation used on our authentication server side, along with a MySQL database to store usernames and accounting information. The whole server is written in C language, and it supports a wide variety of protocols, which includes many EAP authentication methods, such as EAP-TTLS, EAP-TLS, EAP-PEAP, EAP-AKA and EAP-SIM.

Similarly to WPA Supplicant, adding EAP-SH to FreeRADIUS implied creating a C file which implemented the EAP interface methods (such as *instantiate* and *process*), *rlm_eap_<method_name>.c*, and an additional C file which implemented the protocol's logic, *eap_<method_name>.c*.

Also as in WPA Supplicant, in order to have a "head start", we also adapted the freeRADIUS's PEAP modules into our defined protocol architecture.

5.2.1 Configuration

In order to integrate our protocol, a FreeRADIUS development version was downloaded from Github⁷ and the EAP configuration file was changed accordingly, located in the modules folder. On this file, all the following configurations were inserted so to be parsed when the EAP-SH module is initialized:

⁶<http://freeradius.org/>

⁷<https://github.com/FreeRADIUS/freeradius-server/>

```

sh{
    tls{
        certificate_file = /etc/ssl/radiusServerCert.pem
        private_key_file = /etc/ssl/radiusServerCertPrivKey.pem
        private_key_password = "password"
        ca_file = /etc/ssl/certs/CertAuthority.pem
        captive_portal_loc = 192.168.1.4

        staple {
            enable = yes
            override_cert_url = yes
            url = "http://127.0.0.1/ocsp/"
        }
    }
    valid_days = 7
}

```

Code Snippet 6: FreeRADIUS - EAP-SH Configuration

By configuration line:

- **certificate_file**: server certificate sent when authenticating himself through TLS. Contains the certificate chain;
- **private_key_file**: file containing the server certificate private key;
- **private_key_password**: password to decrypt private key file;
- **ca_file**: server CA file, through which it validates the client certificate. Alternatively, instead of a single file, a path can be configured (with **ca_path**);
- **enable (in staple)**: if 'yes', enables OSCP stapling, explained in Section 4.2.3;
- **override_cert_url**: if 'yes', the OSCP Responder URL is extracted from the 'url' field from this file, instead of the default which is to be extracted from the certificate;
- **url**: OSCP Responder URL
- **valid_days**: number of valid days the server signs the client Certificate Signing Request;

5.2.2 Data Structures

EAP-SH has a lot of variables that need to be taken into account when processing a message coming from the supplicant. For that, it possesses two data structures:

```

typedef struct rlm_eap_sh_t {
    fr_tls_conf_t *tls_conf;
    int          valid_days;           //new
    bool         proceed_to_phase2;   //new
    bool         no_certificate;      //new
    bool         tls_start;           //new
} rlm_eap_sh_t;

typedef struct eap_sh_session_t {
    eap_sh_state_t state;             //new
    int            valid_days;        //new

    size_t        record_in_total_len;
    size_t        record_in_rcvd_len;
    bool          record_in_started;
    uint8_t       *record_in;

    size_t        record_out_total_len;
    size_t        record_out_sent_len;
    bool          record_out_started;
    uint8_t       *record_out;

    int*          radius_socket_fd;   //new

    size_t        fragment_size;     //new
} eap_sh_session_t;

```

Code Snippet 7: FreeRADIUS - EAP-SH Data Structures

By variable:

- **tls_conf**: parsed tls configuration from file;
- **valid_days**: parsed valid days configuration from file;
- **proceed_to_phase2**: flag indicating if phase 2 should start;
- **no_certificate**: Flag containing the evaluation of the client's certificate initial validation;
- **tls_start**: Flag indicating when phase 1 (EAP-TLS) should start;
- **state**: State variable that stores the current state of the EAP-SH process on the server (explained further);
- **record_in_total_len**: Total length of the incoming record (payload without flags and length fields);
- **record_in_rcvd_len**: Total fragment length received of the incoming record;

- `record_in_started`: Flag indicating if an incoming record is being received;
- `record_in`: Buffer that stores the incoming record;
- `record_out_total_len`: Total length of the outgoing record (payload without flags and length fields);
- `record_out_rcv_len`: Total fragment length sent of the outgoing record;
- `record_out_started`: Flag indicating if an outgoing record is being sent;
- `record_out`: Buffer that stores the outgoing record;
- `radius_socket_fd`: Socket file descriptor pertaining to the socket connecting to the HTTP server;
- `fragment_size`: Outgoing packet's fragment size;

5.2.3 EAP-SH Server States

Similarly to WPA Supplicant, our protocol also has an internal state machine on the server in order to know how to reply to each received message from the supplicant. The code snippet below lists all the possible states.

```
typedef enum {
    EAP_SH_RECORD_RECV_FIRST,
    EAP_SH_RECORD_RECV_MORE,
    EAP_SH_RECORD_RECV_COMPLETE,
    EAP_SH_RECORD_SENDING,

    EAP_SH_RECV_FIRST_CERT,
    EAP_SH_TLS,
    EAP_SH_START_PHASE2,
    EAP_SH_SENDING_PAGE,
    EAP_SH_PAGES_SENT,
    EAP_SH_RECEIVED_ACK,
    EAP_SH_RECEIVED_REQUEST,
    EAP_SH_RECEIVED_CERT_REQUEST,
    EAP_SH_SENDING_CERTIFICATE,
    EAP_SH_CERTIFICATE_SENT,
    EAP_SH_SUCCESS,
    EAP_SH_FAIL
} eap_sh_state_t;
```

Code Snippet 8: FreeRADIUS - EAP-SH RADIUS States

On the server side, after the initial certificate validation is done (`EAP_SH_RECV_FIRST_CERT`), the server starts an EAP-TLS process (`EAP_SH_TLS`).

Once the TLS tunnel is set, either the process ends successfully (`EAP_SH_SUCCESS`) or the second phase starts (`EAP_SH_START_PHASE2`). On the latter case, the server expects to receive an HTTP GET request and starts sending the captive portal page (`EAP_SH_SENDING_PAGE`), until all page is sent (`EAP_SH_PAGE_SENT`). The authentication server expects one of three messages, at all times (see Section 4.2.1, Figure 4.4):

- Page Acknowledge (`EAP_SH_RECEIVED_ACK`): A fragment acknowledge sent by the supplicant (with all flags set to 0);
- GET/POST Request(`EAP_SH_RECEIVED_REQUEST`): The supplicant can ask for additional HTTP objects or send a POST (G flag is set);
- Certificate Request (`EAP_SH_RECEIVED_CERT_REQUEST`): A certificate signing request is sent by the supplicant (C flag is set).

In response to the last fragment of the captive portal page, the client sends a POST (with G flag set) and the server relays it to the HTTP server. From here, the authentication server sends a result page (`EAP_SH_SENDING_PAGE`) received from the web server, after the latter validates the credentials. As soon as the whole page is sent, and the supplicant is validated successfully (`EAP_SH_PAGES_SENT`), the server receives a Certificate Signing Request (`EAP_SH_RECEIVED_CERT_REQUEST`), signs it and sends it back to the client (`EAP_SH_SENDING_CERTIFICATE` and `EAP_SH_CERTIFICATE_SENT`). The supplicant acknowledges the reception and the server sends an EAP Success message (`EAP_SH_SUCCESS`).

At any time, should the server encounter some error or some inconsistency in any exchanged message, the server sends back an EAP Failure (`EAP_SH_FAIL`).

Fragmentation and defragmentation is processed the same way as the supplicant does, based on the EAP-SH flags. The first three states are used for defragmentation purposes and the fourth state for fragmenting.

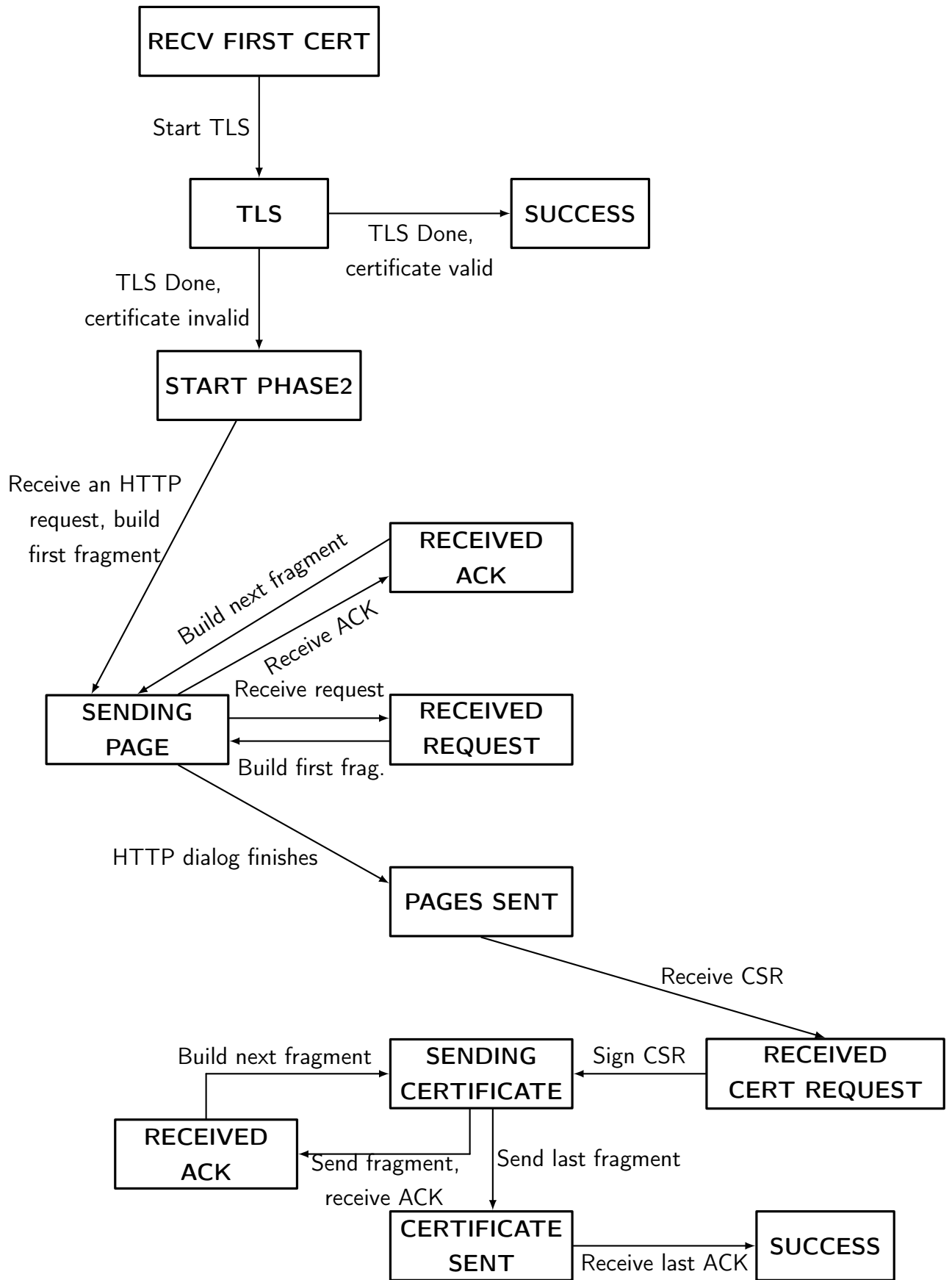
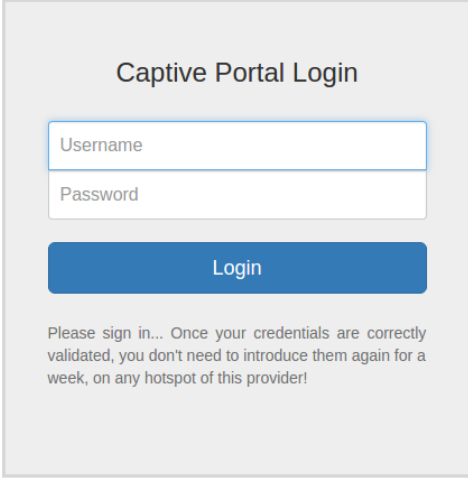


Figure 5.2: EAP-SH: RADIUS State Machine

5.2.4 HTTP Server

The freeRADIUS server is also responsible for communicating directly with the HTTP server, serving as a proxy to the supplicant, decapsulating the HTTP packets, sending them to the web server, receiving responses and encapsulating them in order to send them back to the supplicant. The HTTP server can be located anywhere on the Internet, as a normal server would.

This server was implemented with Apache HTTP Server⁸ and a minimal login page was also implemented, shown in Figure 5.3.



The image shows a web form titled "Captive Portal Login". It features two input fields: "Username" and "Password". Below these fields is a blue "Login" button. At the bottom of the form, there is a line of text: "Please sign in... Once your credentials are correctly validated, you don't need to introduce them again for a week, on any hotspot of this provider!"

Figure 5.3: Implemented Captive Portal

The server also holds a MySQL database to validate received POST messages containing usernames and passwords against it.

⁸<https://httpd.apache.org/>

This chapter intends to give an overall evaluation of the solution's security, which means if the implemented architecture meets the stated objectives of this thesis.

'Instead of focusing on the malware, focus on the attacker. He is greedy, he has weaknesses and he makes mistakes.', Shlomo Toubol

6.1 DOLEV-YAO MODEL CONDITIONS

It is necessary to do a protocol's validation in order to know if it is secure, having in account some realistic conditions. As we've seen before, the EAP-SH method is effective against passive eavesdroppers, who simply monitor the communication and try to decipher messages, since all traffic is encrypted and therefore well-protected. However, we must also consider active eavesdroppers, who may be able to impersonate an entity and may alter or replay messages. For this security analysis, we will assume the Dolev-Yao Model [17] conditions. For that, we must initially make a few assumptions:

1. The public key system is perfect:
 - Hash functions are unbreakable (non-invertible, collision-free);
 - The public directory is secure and cannot be tampered;
 - Everyone has access to all E_x , *i.e.*, all the messages encrypted with entity X public key;
 - Only entity X knows D_x , *i.e.*, only X knows how to decipher those messages.
2. Only the two entities who wish to communicate are involved in the transmission process (encryption and decryption);
3. The same format is used for any pair of entities that wish to communicate;
4. The attacker has control over the entire network:
 - He can obtain any message passing through the network;
 - He is a legitimate user of the network, can initiate a conversation with any party, or be a receiver to any party;

6.1.1 Rogue Access Points

A rogue access point is a wireless AP installed on a secure network without any authorization from that network's administrator. This type of AP does not constitute a threat, and as it only serves as a middleman it can only observe the traffic and is not able to decrypt it. Furthermore, the AP can direct all the traffic to a malicious server, that simulates the authentication server behaviour, but again the server must be authenticated through TLS and fails to do so if it is a fraud. A rogue AP also cannot communicate with the genuine authentication server because normally this communication involves a shared secret that the rogue AP does not know.

6.1.2 Client Certificate Owner

The attacker may try to pass as the owner of the transmitted client (supplicant) certificate, which can be used to obtain a valid authentication. For that, EAP-TLS (EAP-SH phase 1) has a mechanism for the client to prove to the server that he possesses the private key corresponding to its public key certificate. In other words, that he is the owner of the sent certificate. The Certificate Verify message is a hash of all the messages up to the point where the client sends its certificate and it is encrypted with the client's private key. The server verifies this signature with the public key, which ensures that it was signed with the client's private key. Through this mechanism, it is impossible for the attacker to impersonate the client just by stealing his public key certificate.

6.1.3 Server Certificate Owner

Server's impersonation may be one way of obtaining the credentials from a client. For this, an attacker might try to pass as the owner of the server certificate. In EAP-TLS, the server can only generate the master secret if it decrypts a random secret sent by the client and encrypted with the server's private key. Only the legitimate server has the private key and so only this server can generate the master secret (and its client). Therefore, the server's impersonator is not able to decrypt or encrypt data on the resulting TLS tunnel.

6.1.4 Client Identity

In EAP-TLS, identity is generally sent without any protection and the attacker may snoop on the identity, or even modify or spoof identity exchanges. To address these threats, the client certificate sent to the server contains on its Subject Alternative Name the identity (used for accounting purposes), encrypted with the server's public

key. When the attacker captures the client certificate, he is not able to discover the client's identity, because only the server can decrypt it.

6.1.5 Captive Portal Circumvention

Normally an attacker who would like abusing the network would take advantage of opened protocols in a hotspot environment. A very good example is encapsulating TCP inside protocols such as DNS or ICMP. This implementation does not leave any protocol prone to such abuse since the supplicant, before and during the authentication, does not receive an IP address and so cannot tunnel any desired protocol. Furthermore, the AP blocks all arbitrary communications with the network until a successful end of the EAP-SH and a posterior four-way handshake.

6.1.6 Captive Portal Impersonation

An attacker can impersonate a captive portal in order to obtain a client's credentials. On this case, this attacker simulates, with a false AP, a captive portal similar to the one a client uses when performing EAP-SH. Having that in account:

1. The supplicant must indicate that authentication is to be performed with EAP-SH, refusing other hotspots without any security whatsoever;
2. The authentication's server certificate is validated by the supplicant, preventing therefore that the supplicant communicates directly with the false captive portal.

Seven

Conclusion

This chapter presents a final conclusion about this thesis, describing the problem, the solution and what is left as future work.

'It's more fun to arrive a conclusion than to justify it.', Malcolm Forbes

Public hotspots are potentially insecure networks found in just about any minimally crowded place, such as shopping centres. Their usage increases every day, along with their subscribers, who need to authenticate themselves in order to use network resources. This type of authentication is done through a captive portal, which is a web page where the user proves he is a subscriber by logging in using a username-password pair. Their insecurity poses a threat as sensitive information circulate sometimes with no confidentiality nor integrity control, among other problems specified in Sections 1.1 and 2.3.2.

The solution that we developed, EAP-SH, intends to take advantage of the 802.1X architecture, and consists on the implementation of an EAP method (for the second phase of 802.1X authentication) complemented with the captive portal paradigm. This EAP method is performed in two stages: first the establishment of a TLS tunnel; then an HTTP-based authentication via a captive portal. Most of the advantages that this method provides repress the weaknesses of current hotspots' architecture:

1. The first phase of the protocol allows mutual authentication, proving the legitimacy of network and possibly the client, in case his certificate is successfully validated;
2. HTTP-based authentication is already familiar to users (logging into a captive portal);
3. The establishment of a TLS tunnel allows exchanged data, including credentials, to be encrypted and its integrity assured;
4. An installed client certificate, as a result of the process, may be valid for a defined period of time. This means that for that same time, the client does not need to perform HTTP-based authentications;
5. AP software modification is not needed;

6. Harmful clients cannot abuse the network through the tunnelling of protocols.

As future work, we leave the study of how to adapt current captive portals' in various scenarios (such as stores, airports, shopping centres, cafés, domestic operator installations) into the exploration of EAP-SH. Also, it might be possible a future publication of EAP-SH as an RFC.

References

- [1] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, and H. Levkowitz, *Extensible Authentication Protocol (EAP)*, RFC 3748 (Proposed Standard), Updated by RFCs 5247, 7057, Internet Engineering Task Force, Jun. 2004. [Online]. Available: <http://www.ietf.org/rfc/rfc3748.txt>.
- [2] «Ieee standard for local and metropolitan area networks—port-based network access control», *IEEE Std 802.1X-2010 (Revision of IEEE Std 802.1X-2004)*, pp. 1–205, Feb. 2010. DOI: [10.1109/IEEESTD.2010.5409813](https://doi.org/10.1109/IEEESTD.2010.5409813).
- [3] W. Simpson, *The Point-to-Point Protocol (PPP)*, RFC 1661 (INTERNET STANDARD), Updated by RFC 2153, Internet Engineering Task Force, Jul. 1994. [Online]. Available: <http://www.ietf.org/rfc/rfc1661.txt>.
- [4] A. Zúquete, *Segurança em redes informáticas*, FCA, Ed. Lidel, 2013, ISBN: 9789727227679.
- [5] S. Bellovin and R. Housley, *Guidelines for Cryptographic Key Management*, RFC 4107 (Best Current Practice), Internet Engineering Task Force, Jun. 2005. [Online]. Available: <http://www.ietf.org/rfc/rfc4107.txt>.
- [6] D. Simon, B. Aboba, and R. Hurst, *The EAP-TLS Authentication Protocol*, RFC 5216 (Proposed Standard), Internet Engineering Task Force, Mar. 2008. [Online]. Available: <http://www.ietf.org/rfc/rfc5216.txt>.
- [7] P. Funk and S. Blake-Wilson, *Extensible Authentication Protocol Tunneled Transport Layer Security Authenticated Protocol Version 0 (EAP-TLSv0)*, RFC 5281 (Informational), Internet Engineering Task Force, Aug. 2008. [Online]. Available: <http://www.ietf.org/rfc/rfc5281.txt>.
- [8] A. Palekar, S. Josefsson, D. Simon, and G. Zorn, «Protected EAP Protocol (PEAP) Version 2», Internet Engineering Task Force, Internet-Draft draft-josefsson-pppext-eap-tls-eap-10, Oct. 21, 2004, Work in Progress, 87 pp. [Online]. Available: <https://tools.ietf.org/html/draft-josefsson-pppext-eap-tls-eap-10>.
- [9] H. Haverinen and J. Salowey, *Extensible Authentication Protocol Method for Global System for Mobile Communications (GSM) Subscriber Identity Modules (EAP-SIM)*, RFC 4186 (Informational), Internet Engineering Task Force, Jan. 2006. [Online]. Available: <http://www.ietf.org/rfc/rfc4186.txt>.
- [10] J. Arkko and H. Haverinen, *Extensible Authentication Protocol Method for 3rd Generation Authentication and Key Agreement (EAP-AKA)*, RFC 4187 (Informational), Updated by RFC 5448, Internet Engineering Task Force, Jan. 2006. [Online]. Available: <http://www.ietf.org/rfc/rfc4187.txt>.
- [11] K. J. Hole, E. Dyrnes, and P. Thorsheim, «Securing wi-fi networks», *Computer*, vol. 38, no. 7, pp. 28–34, Jul. 2005, ISSN: 0018-9162. DOI: [10.1109/MC.2005.241](https://doi.org/10.1109/MC.2005.241).
- [12] C. Gentry and A. Silverberg, «Hierarchical id-based cryptography», in *Advances in Cryptology — ASIACRYPT 2002: 8th International Conference on the Theory and Application of Cryptology*

and *Information Security Queenstown, New Zealand, December 1–5, 2002 Proceedings*, Y. Zheng, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 548–566, ISBN: 978-3-540-36178-7. DOI: 10.1007/3-540-36178-2_34. [Online]. Available: http://dx.doi.org/10.1007/3-540-36178-2_34.

- [13] J. Choi, S. Y. Chang, D. Ko, and Y. C. Hu, «Secure mac-layer protocol for captive portals in wireless hotspots», in *2011 IEEE International Conference on Communications (ICC)*, Jun. 2011, pp. 1–5. DOI: 10.1109/icc.2011.5963508.
- [14] S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams, *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP*, RFC 6960 (Proposed Standard), Internet Engineering Task Force, Jun. 2013. [Online]. Available: <http://www.ietf.org/rfc/rfc6960.txt>.
- [15] Y. Pettersen, *The Transport Layer Security (TLS) Multiple Certificate Status Request Extension*, RFC 6961 (Proposed Standard), Internet Engineering Task Force, Jun. 2013. [Online]. Available: <http://www.ietf.org/rfc/rfc6961.txt>.
- [16] C. Rigney, S. Willens, A. Rubens, and W. Simpson, *Remote Authentication Dial In User Service (RADIUS)*, RFC 2865 (Draft Standard), Updated by RFCs 2868, 3575, 5080, 6929, Internet Engineering Task Force, Jun. 2000. [Online]. Available: <http://www.ietf.org/rfc/rfc2865.txt>.
- [17] D. Dolev and A. Yao, «On the security of public key protocols», *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–208, Mar. 1983, ISSN: 0018-9448. DOI: 10.1109/TIT.1983.1056650.