Department of Computer Science

Secure configuration and management of connected devices

Credential Provisioning and Peer Configuration with Extensible Authentication Protocol

---

Sebastien Boire

Aalto University

Tartu University

MASTER THESIS

Aalto University, Tartu University

MASTER THESIS 2021

Secure Configuration and Management of Internet of Things Devices

Credential Provisioning and Peer Configuration with Extensible Authentication Protocol

Sebastien Boire

Thesis submitted for examination for the degree of Master of Computer Science.

Otaniemi, 14 May 2021

Supervisors: professor Tuomas Aura, professor Dominique Unruh
Advisor: Philip Ginzboorg, D.Sc. (Tech.),
Sandeep Tamrakar, D.Sc. (Tech.)

Aalto University
School of Science
Department of Computer Science

Tartu University
Institute of Computer Science

Aalto University

Aalto University, P.O. Box 11000, FI-00076 Aalto www.aalto.fi

Tartu University, Ülikooli 18, 50090 Tartu www.ut.ee

Abstract

Author Sebastien Boire

Title Credential Provisioning and Peer Configuration with EAP

School School of Science

Department Department of Computer Science

Research field Computer Science

Level master thesis **Date** 14 May 2021 **Pages** 67 **Language** English

Abstract

The Internet of Things (IoT) contains an increasing number of diverse objects, ranging from simple sensors to smart speakers and industrial appliances. The continuing growth in the number and the diversity of connected devices within enterprises and homes complicates their management. Vendor-specific protocols cannot solve this problem.

The Extensible Authentication Protocol (EAP) is a framework to negotiate and run EAP methods, i.e. authentication protocols between client and server. Tens of different EAP methods exist, and EAP is widely-adopted in WiFi and cellular networks. In some EAP methods the server can invoke another, "inner" EAP method for additional authentication inside the same EAP session.

In this thesis we investigate how to apply EAP for managing devices in wireless networks. Our approach is to add the possibility to send short client tokens from server to client in EAP session. After successful authentication and completion of the EAP session, the client uses these tokens to access the management servers.

We have designed several options for transferring client tokens inside an EAP session. These options were then implemented by extending open-source software components and evaluated experimentally, using Raspberry Pi as a platform.

Based on our analysis and experiments, the most flexible option for sending client tokens in EAP is by combination of an outer EAP method (EAP-oPROV) that sequentially runs two inner EAP methods. The first inner method does peer authentication, and the tokens are sent to the client in the second inner EAP method (EAP-iPROV). Since the first inner EAP method is not fixed (it is chosen by the authentication server), there are many compatible EAP methods for peer authentication in this option. The two new EAP methods (EAP-oPROV and EAP-iPROV) could be standardized in the future.

Keywords IoT, EAP, authentication, credentials, certificate

URL https://github.com/Sebastien2/EAP_device_lifecycle

Preface

I want to thank professor Tuomas Aura and professor Dominique Unruh, my instructors Philip Ginzboorg and Sandeep Tamrakar and my colleagues Tolgahan Akgün, Pekka Laitinen and Sampo Sovio.

Sebastien Boire

Contents

1. Introduction

With the increasing number and diversity of Internet of Things (IoT) devices, the configuration and maintenance of the devices in WiFi networks are becoming more expensive and complex. The diversity of the IoT devices is reflected in the many management protocols used by different vendors.

Authentication, provisioning and configuration operations may be repeated many times during the lifecycle of the device. These operations occur for the first time when a new device joins a wireless network. Authentication verifies the identity of the device; provisioning equips the device with long-term credentials; configuration defines the parameters necessary to the correct working of the device. Provisioning and configuration can only take place after authentication.

We consider management of a device's lifecycle in two settings. The first is a large WiFi network, for instance, a university campus, a company office, or a government building. In this use case, it would be beneficial to have a unified way for configuring all devices. In addition, the devices can belong to different users: the management of credentials requires a dedicated and unified method. The devices also have to be provided with specific rights: depending on their owners and their function, they may have different access rights granted in the provisioning step. The second setting is a home WiFi network. Although the number of devices is not large, the authentication and configuration of each device may be cumbersome for the home user.

In wireless networks, especially in large WiFi networks, authentication is often performed with the Extensible Authentication Protocol (EAP), which includes a diversity of authentication methods for different applications. The EAP authentication framework is specified in RFC 3748 [1]. It has been extended with more than fifty different authentication methods like EAP-TEAP (Tunneled EAP) [2], EAP-TTLS (Tunneled Transport Layer Security) [3] or EAP-FAST (Flexible Authentication via Secure Tunneling) [4]. When a new device wants to connect to a WiFi network, EAP is used before wireless connectivity is established between the device and the WiFi AP (Access Point). The type of authentication, e.g., whether it is mutual or one-sided, depends on the EAP method and on network settings. The three entities participating in EAP

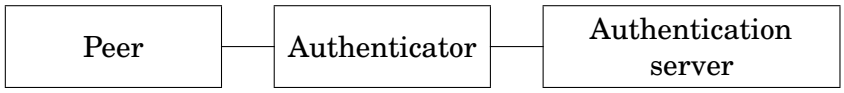

Figure 1.1. Entities in the authentication of a peer

session are shown in Figure 1.1.

The *peer* – also called supplicant in [5] – is the device attempting to access the network, the *authenticator* is the WiFi provider, and the *authentication server* is a server or online service which can verify the identity of the peer. The peer initially has no IP address and therefore is limited to communicating with the authenticator. The WiFi provider transfers the authentication communication to the authentication server. If the authentication server accepts the peer's identity, the authenticator will allow the peer to connect to the network.

In addition to the authentication of new device, it is typically necessary to also perform provisioning and configuration. These three operations – initial authentication, provisioning and configuration – are often called bootstrapping or on-boarding of the device. Currently, EAP does not include a generic method for the configuration and provisioning of the connected device. As a result of the provisioning operation, the device gets an identity and long-term credentials for subsequent network or service access, e.g., a certificate signed by a certification authority. As a result of the configuration operation, the device gets the necessary settings for its correct functioning; for example, it may receive a firmware update.

Two EAP methods are especially relevant to our work: EAP-TEAP (Tunneled Extensible Authentication Protocol) RFC [2] and EAP-CREDS (Credentials Provisioning and Management) internet draft [6]. EAP-TEAP includes optional provisioning of certificate to the peer. However, the support for provisioning is restricted to one EAP method. The EAP-CREDS draft proposal defines a way to perform provisioning and configuration within an external EAP method (e.g., within EAP-TEAP). The proposal has broad goals, including support for several provisioning protocols and arbitrary length of provisioned data. As a result, EAP-CREDS may be difficult to implement and test. One disadvantage of doing provisioning and configuration entirely inside an EAP session is the increase of the authentication server's load.

In this thesis we investigate a different approach. Only the information that is needed to initialize provisioning and configuration is exchanged through an EAP method. This information includes the URLs and access tokens of the provisioning and configuration servers. A successful completion of the EAP session (i.e., successful authentication) triggers provisioning and configuration of the device over HTTPS. One advantage of this approach is that there is no need to send large messages inside EAP session, or include the potentially heavy management operations inside EAP session.

The rest of the thesis is organized as follows. Chapter 2 provides an overview of networking technologies that are relevant for our work. Chapter 3 describes the requirements. Chapter 4 presents the design choices studied in this thesis for adding provisioning and configuration to EAP, and chapter 5 describes the implementation and performance evaluation of these choices. In chapter 6 we analyze and discuss our results and also outline directions for future work. Chapter 7 concludes the thesis.

2. Motivation and Background

In this chapter we first give a model of device lifecycle and identify the specific state in that model on which this work focuses. Second, we outline three types of networks where the EAP-based provisioning and configuration solution could be deployed. Finally, we provide an overview of the Extensible Authentication Protocol and outline four EAP methods: EAP-TLS, EAP-TTLS, EAP-TEAP and EAP-CREDS that are especially relevant to our work. Information about additional EAP methods can be found in Appendix B.

2.1 Device lifecycle

In [7], the lifecycle of an IoT device is defined as having two modes: factory and operational. In the factory mode, the device is built by the manufacturer and provisioned with initial credentials. These credentials may include a certificate signed by the manufacturer, and a list of trusted peers. A device in factory mode can perform bootstrapping and move to operational mode.

In operational mode, the device communicates with a controller using its operational credentials. In Figure 2.1, the bootstrapping operation moves the device into operational mode, and factor reset operation moves the device back to factory mode. Factory reset is needed, for example, when the device changes owner: credentials used by one user should not be available to the following user.

The bootstrapping procedure can be divided into authentication, provisioning of long-term credentials and configuration of the device. In this thesis, we consider an additional state

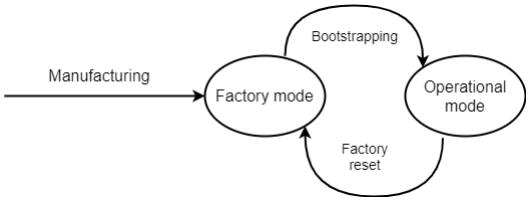

Figure 2.1. Lifecycle of an IoT device (adapted from [7]).

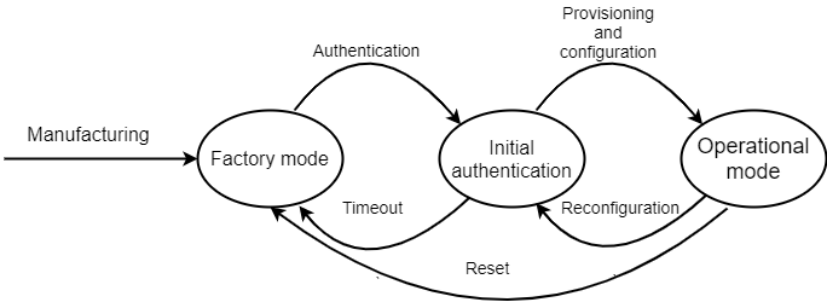

Figure 2.2. Device lifecycle modes.

between authentication and provisioning operations. It is shown in Figure 2.2 as an additional Initial Authentication (IA) mode. A device in the IA mode has been authenticated but not yet provisioned or configured. This additional mode allows the device to be reconfigured several times without re-authentication. A device in IA mode returns back to factory after a timeout, unless it has moved to operational mode.

The next sections provide a device management scenario more efficient in number of required communications, and easier for the end user. Although the additional mode allows wider application, a separation is required to securely manage the device: in the mode IA, the device should not have access to all network resources. The network separation may be necessary to protect from devices which have not yet entirely been initialized.

Configuration determines the purpose of the device. Once configured, the device may be required to provide a proof of the software it executes (called device attestation): this verification justifies the separation of network.

2.2 Deployment environments

Our solution for provisioning and configuration of IoT devices can be potentially deployed in (at least) three types of WiFi networks: campus, home and UAVs (Unmanned Aerial Vehicles).

2.2.1 Campus network

The campus WiFi network could be deployed in the premises of an enterprise, like Huawei office, or a public institution, like Aalto University. The number of devices in the campus network is large; it could be in the hundreds or even thousands. The network includes many devices with multiple owners and EAP is typically used for device authentication. Bootstrapping of a large number of new devices in a campus network could be an issue due to their multitude and variety.

One approach to solve this issue is to define a specific area (zone) for bootstrapping in the campus, and do initial authentication of new devices in that area. As a result of the initial

authentication, the device will get short-term credentials. Afterwards the device will be deployed in the campus, and it will use its short-term credentials to obtain long-term credentials and other configuration information. In this way, the device can be provisioned and configured anywhere in the campus.

Eduroam [8] is an example of EAP-based roaming between campus networks. In Eduroam, the authenticator determines the authentication server to connect to upon reception of a first EAP message by looking at the realm part of the username in the client's first message. Suppose, for example, that the device having peer identity "sebastien.boire@aalto.fi" wants to connect to the WiFi network of Ecole Polytechnique campus in Paris. From the realm part "aalto.fi", the authenticator in the campus network routes the EAP messages to the correct authentication server – in the example, the Aalto University authentication server in Finland. Upon successful authentication, the authenticator provides network access to the peer.

2.2.2 Home network

Today's home WiFi networks are characterized by a relatively small number (e.g., about 10) of IoT devices and users whose technical knowledge varies a lot. A bootstrapping procedure in home environment typically involves the user, but as little as necessary user's intervention should be required. Referring to the lifecycle model in Figure 2.2, the user is required only to perform authentication in order for the device to reach the Initial Authentication (IA) state: the user provides the credential with his action. This can be done with an OOB (Out-Of-Band) channel, for instance using EAP-NOOB [9]. Once in IA state, the device can do provisioning and configuration without human intervention thanks to initial information provided during authentication. The device can be updated through regular configuration without human presence.

Even though EAP is not common in home networks today, as the number and variety of home WiFi devices increase, it is plausible that some device management mechanisms from enterprise networks will be taken into use also at home. For example, future home network could connect to the internet via a gateway device configured with WPA-Enterprise standard mechanisms.

2.2.3 Unmanned Aerial Vehicle network

There is an increasing interest in networks of Unmanned Aerial Vehicles (UAVs): multiple applications are considered in civilian and military areas. In [10], the possible applications are analyzed with the corresponding challenges. Typically, the UAVs form a WiFi mesh network, and an addition of a new UAV to that network (bootstrapping) is done via a dedicated access point (AP) in one of the UAVs. Also the communication between the UAVs to the ground station

is routed via a dedicated AP.

Since communication to the ground station takes longer and consumes more energy than communication with the neighbouring UAVs, it is advantageous to perform only the initial authentication of new UAV with the ground station, and subsequently do provisioning and configuration with other UAVs. Please note that initial authentication of a new UAV with the ground station could be potentially done with EAP.

2.3 Methods for first authentication of the device

The first authentication of the device takes place before any provisioning. Therefore, the device may be unidentified at that time. First authentication relies on specific mechanisms. In [11], first authentication methods are classified in four groups:

- Managed methods: there is a pre-existing credential in the device, which is used for first authentication. This first credential may have been provided by the manufacturer.
- Ad-hoc methods: first authentication is executed using an Out-Of-Band (OOB) channel. For instance, EAP-NOOB identifies the client by its physical closeness to the controller. Other OOB channels include sound, ultrasound, a QR code or a button on the client.
- Leap-of-faith methods: these methods assume the first connection was honest. Future authentication will verify that the device has the same identity as on first authentication.
- Hybrid methods: the combination of managed and ad-hoc methods is the most commonly used.

2.4 Extensible Authentication Protocol (EAP)

EAP [1] is a generic framework for transporting messages of different authentication methods, called EAP *methods*. It is a request-response protocol: for example, the peer (client) answers with a response to the authentication server's request, and then the server can send a new request. EAP can operate directly on top of wireless LAN or PPP (Point-to-Point) protocol [12] without establishing an IP connection. EAP is often used in wireless networks to authenticate the client before granting network access.

EAP does not natively provide fragmentation support: one message cannot be split in several packets. Therefore, the size of one message is limited by the MTU (Maximum Transfer Unit) on the link. However, some EAP methods can provide fragmentation support (for instance, methods exchanging certificates like EAP-TTLS).

EAP packets contain the following attributes:

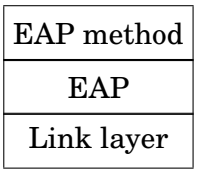

Figure 2.3. EAP protocol stack

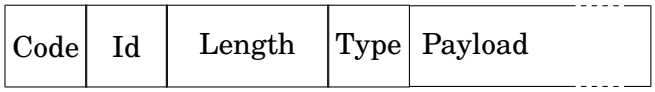

Figure 2.4. Format of an EAP packet

- Code (one byte): indicates whether it is a request or a response.
- Identifier (one byte): helps associate a response to a request.
- Length (two bytes).
- Type (one byte): determines the nature of the payload. Some of the types are Identity, Notification, One Time Password, Expanded Types.
- Payload.

The payload is specific to the type of the packet. Some types are described in figure 2.1.

Type code	type name
1	identity, this communicates an unprotected identifier [1]
2	notification, this allows the transfer of arbitrary data [1]
3	NaK, this is a response to an unacceptable proposed method [1]
4	MD5-challenge [1]
5	OTP (One Time Password) [1]
6	GTC (Generic Token Card) [1]
13	EAP-TLS authentication protocol [13]
17	Cisco-LEAP
18	EAP-SIM [14]
21	EAP-TTLS [3]
25	PEAP [15]
43	EAP-FAST [4]

Table 2.1. Type codes of widespread EAP methods

2.4.1 EAP-TLS

EAP-TLS (Transport Layer Security) [13] defines the execution of a TLS handshake [16] within EAP. It may be run as EAP method when the client already has a certificate: the server can request client authentication within the TLS handshake with a certificate.

EAP-TLS provides session resumption, fragmentation and sharing of key material.

Although EAP-TLS provides strong security, it requires the distribution of certificates to the client, which could be costly in a large network.

2.4.2 EAP-TTLS

EAP Tunneled Transport Layer Security (EAP-TTLS) [3] is a method allowing authentication of the server, secure encrypted communication, and authentication of the client. It operates in two phases:

- Phase one: the server is authenticated by the client, and they exchange materials for an encryption key. At the end of phase one, a TLS channel has been established between the client and the authentication server.
- Phase two: the client authenticates the server, and the client may authenticate the server depending on the phase two algorithm. This step is optional.

Unlike EAP-TLS, the client certificate is optional: it is therefore easier to use in large networks.

Figure 2.5 describes the execution of EAP-TTLS with the messages transferred between each of the three entities taking part.

2.4.3 EAP-TEAP

EAP Tunneled Extensible Authentication Protocol (EAP-TEAP)[2] is an EAP method widely used in wireless networks. It allows mutual authentication, secure encrypted channel and provisioning of certificate to the client. EAP-TEAP operates in two phases, both being mandatory:

- Phase one: a TLS handshake allows server authentication and setting of a secure encrypted tunnel.
- Phase two: additional information is exchanged in the encrypted tunnel. This can include client authentication.

EAP-TEAP uses TLV objects in phase two.

EAP-TEAP supports the execution of an EAP-method within phase two: internal EAP packets are wrapped in TLV objects used by EAP-TEAP in phase two, as an inner EAP method. EAP-CREDS can be executed at this level of EAP-TEAP. It is also possible to perform client

authentication and client provisioning in phase two. Multiple internal protocols can be executed sequentially in phase two.

The provisioning of the peer can be performed via a certificate or via a Protected Access Credential (PAC). A PAC contains a key and a field used for long-term authentication to the authentication server. The development of provisioning motivated the evolution of EAP-FAST [4] into EAP-TEAP. A PAC allows fast re-authentication over EAP, wheas a certificate can be used either as long term credential for the client, or to learn the identity of the local authority. This evolution supports the interest of executing provisioning inside EAP.

The execution of an internal EAP method is performed with the TLV type EAP-Payload-TLV.

Figure 2.6 represents the EAP-TEAP communication with its two phases.

However, EAP-TEAP is one specific EAP method: the peer may not support this EAP method, resulting in incompatibility. In addition, the configuration is not performed in EAP-TEAP. Since the configuration data may be quite big, it may be impossible to add in phase two of EAP-TEAP.

2.4.4 EAP-CREDS

EAP Credentials Provisioning and Management (CREDS) [6] [17] is a specification draft for secure provisioning and configuration of the client through EAP.

EAP-CREDS operates within several EAP methods, like EAP-TLS or EAP-TEAP. EAP-CREDS focuses on the communication of credentials after the establishment of a secure authenticated channel. EAP-CREDS includes three phases:

- Phase one: initialization. This step selects the credential management protocol.
- Phase two: provisioning (optional). The two parties exchange the credential data. These messages include two parts: the provisioning headers and the provisioning data.
- Phase three: validation (optional). This step allows the authenticator to verify that the credentials have been correctly received and interpreted by the peer.

EAP-CREDS uses TLV objects to transport the credentials. Most of credential types are supported: X.509 certificate, public key, symmetric key, username and password, and one-time password are compatible with EAP-CREDS.

With EAP-CREDS, it is possible to perform authentication, provisioning and configuration within EAP protocol. This implies that a single server performs these three tasks. Although EAP-CREDS is a possible solution to provisioning and configuration, it relies on fragmentation within EAP in order to send long messages. EAP is not adapted to send long information –

e.g. configuration information. Therefore, it would be interesting to limit EAP to sending the minimal required information and then configure the client with another protocol.

2.5 WiFi Protected Access

WiFi Protected Access (WPA) is a security certification program defined by the WiFi Alliance standard. WPA and its successors (WPA2 and WPA3) provide sets of protocols that satisfy security requirements. The security requirements protect from known attacks on connection to a wireless network.

2.5.1 Network categories

WPA separates networks in two categories. WPA-Personal focuses on home and small professional networks. It does not require an authentication server and relies on WPA-PSK (pre-shared key): the client connects to the authenticator using a password, from which an encryption key is derived. WPA-Enterprise addresses enterprise networks and requires an authentication server.

2.5.2 EAP support on WPA-Enterprise

WPA-Enterprise supports the following EAP methods:EAP-TLS, EAP-TTLS/MSCHAPv2, EAP-PEAP/MSCHAPv2, EAP-PEAP/GTC (Generic Token Card), EAP-PEAP/TLS, EAP-SIM, EAP-AKA, EAP-FAST. The choice of method depends on network configuration.

EAP-SIM and EAP-AKA rely on a SIM module in the client for authentication. All other EAP methods listed perform a TLS handshake to obtain a secure communication tunnel.

Enterprises that require high security in their internal network commonly use EAP-TLS where client certificates are mandatory: all devices must have a certificate to access the network. (Please note that it is possible to use client certificates also in other TLS-based EAP methods.) In universities and open environments, the distribution of device certificates for EAP-TLS is too restrictive.

2.5.3 WiFi connection establishment

A WiFi connection is initiated with the EAPOL (EAP over LAN) protocol [5]. Depending on its configuration, EAPOL does not always call EAP. For instance, in a WPA-Personal network, EAPOL does not authenticate the client, and only generates an encryption key from the password. The connection establishment is represented on Figure 2.8.

When a user manages IoT devices in his home via a cloud platform (e.g., Amazon's IoT Core AWS), the EAP authenticator can be located in the home gateway, and the authentication

server in a the cloud platform.

In WPA-Enterprise, EAP is executed before the exchange of EAPOL Key messages. With the execution of an EAP method, the EAPOL messages are encrypted using the key materials generated by EAP. An example of a WPA-Enterprise protocol where the authentication is done using EAP-PWD is shown in Figure 2.9.

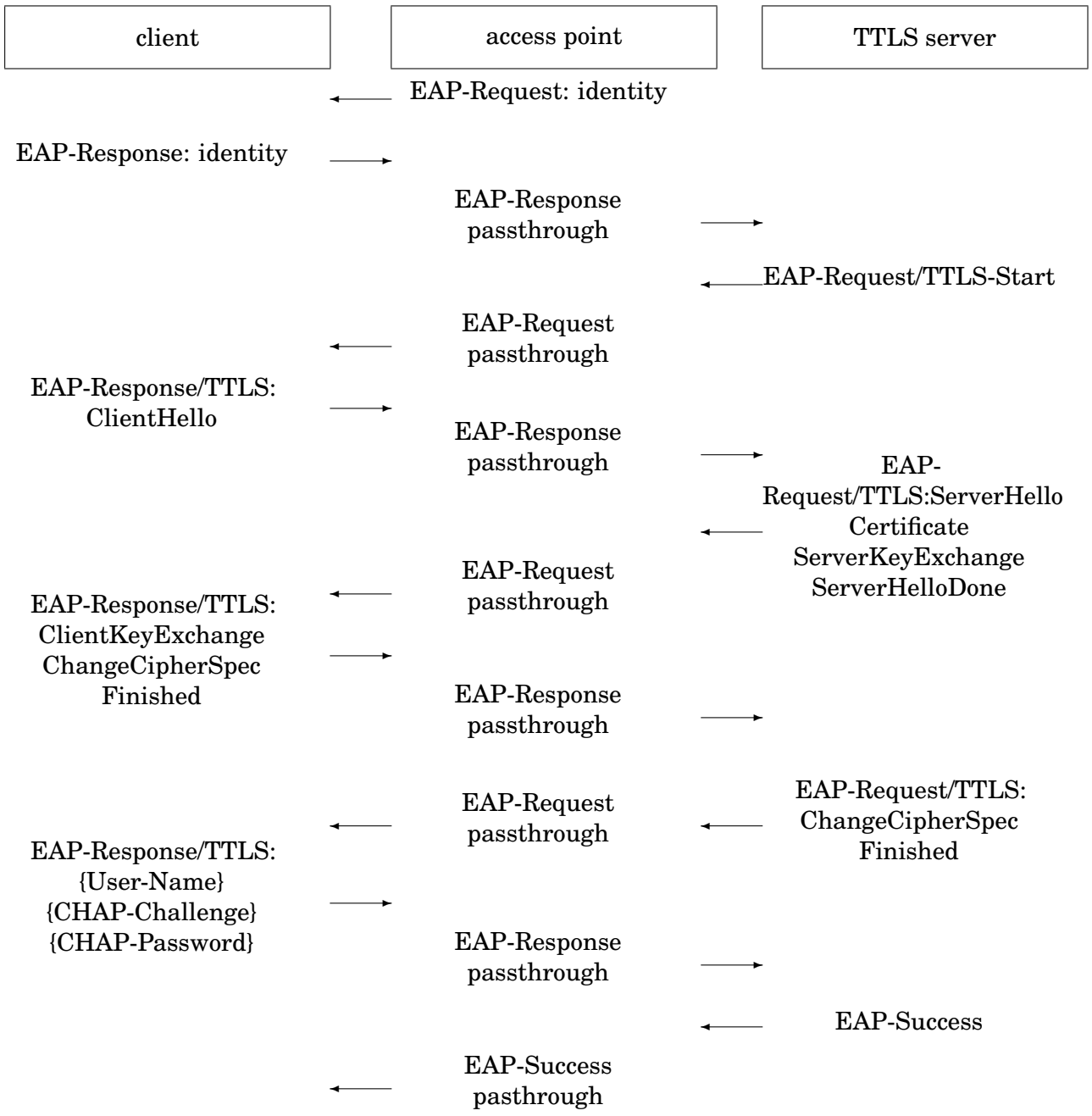

Figure 2.5. EAP-TTLS successful message exchange, with CHAP as client authentication method

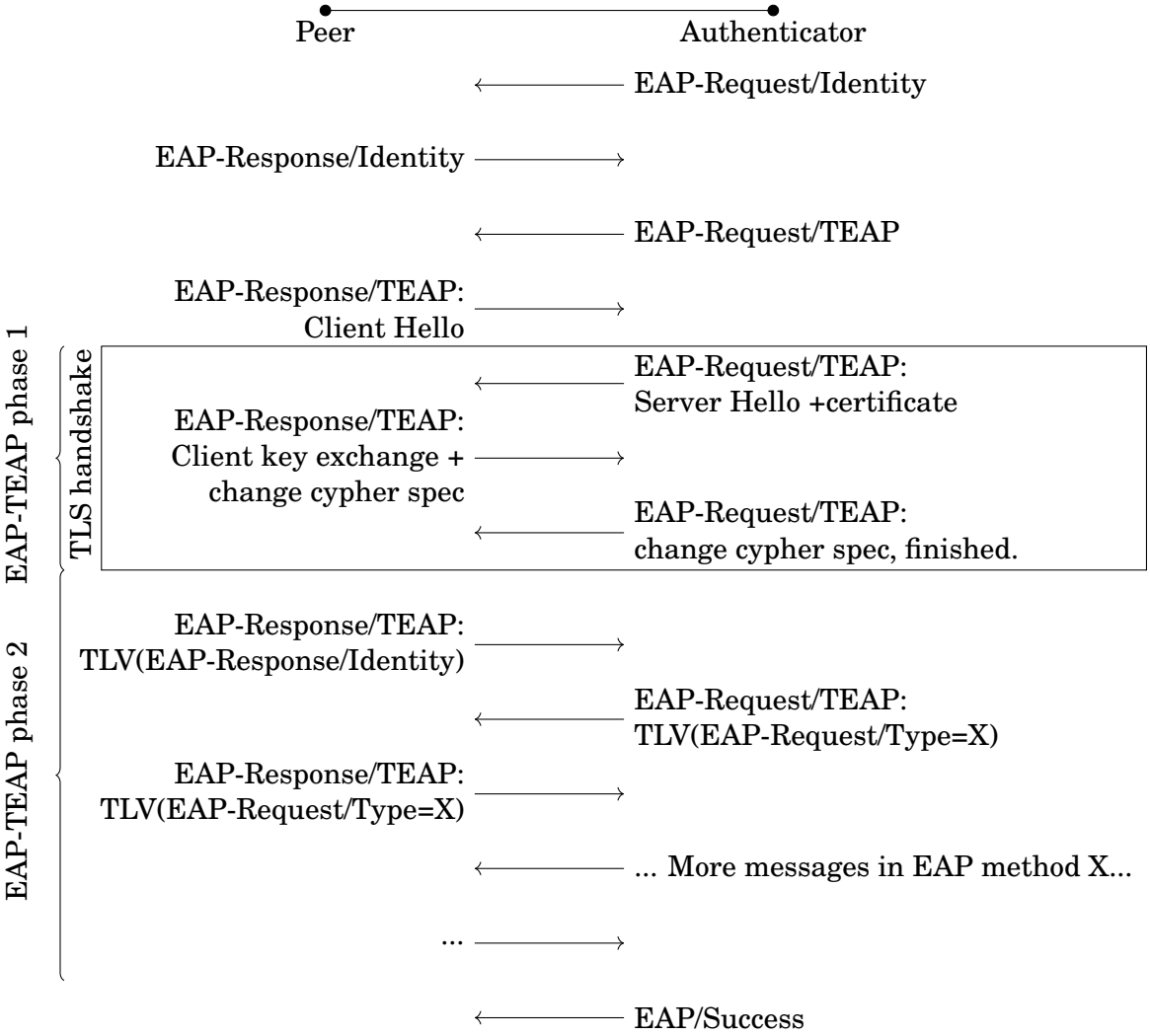

Figure 2.6. Messages in EAP-TEAP with internal EAP method in phase 2

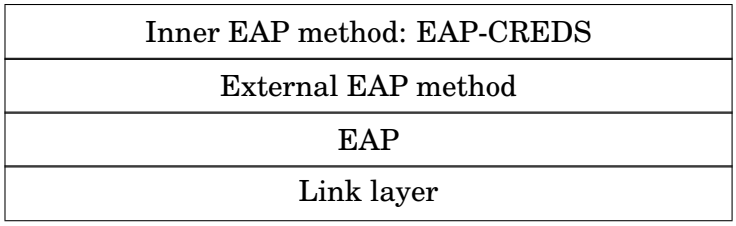

Figure 2.7. EAP-CREDS protocol stack

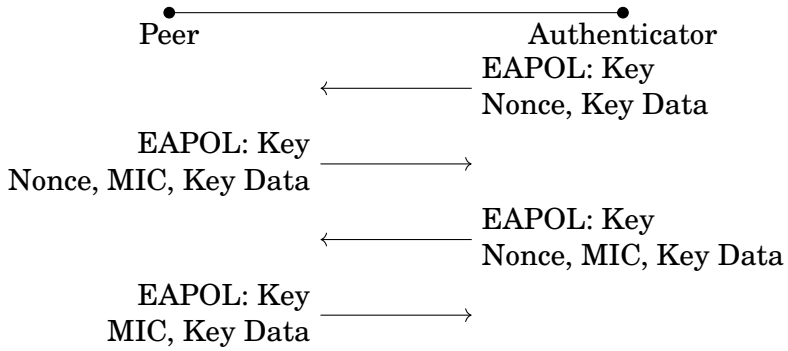

Figure 2.8. WPA2-Personal

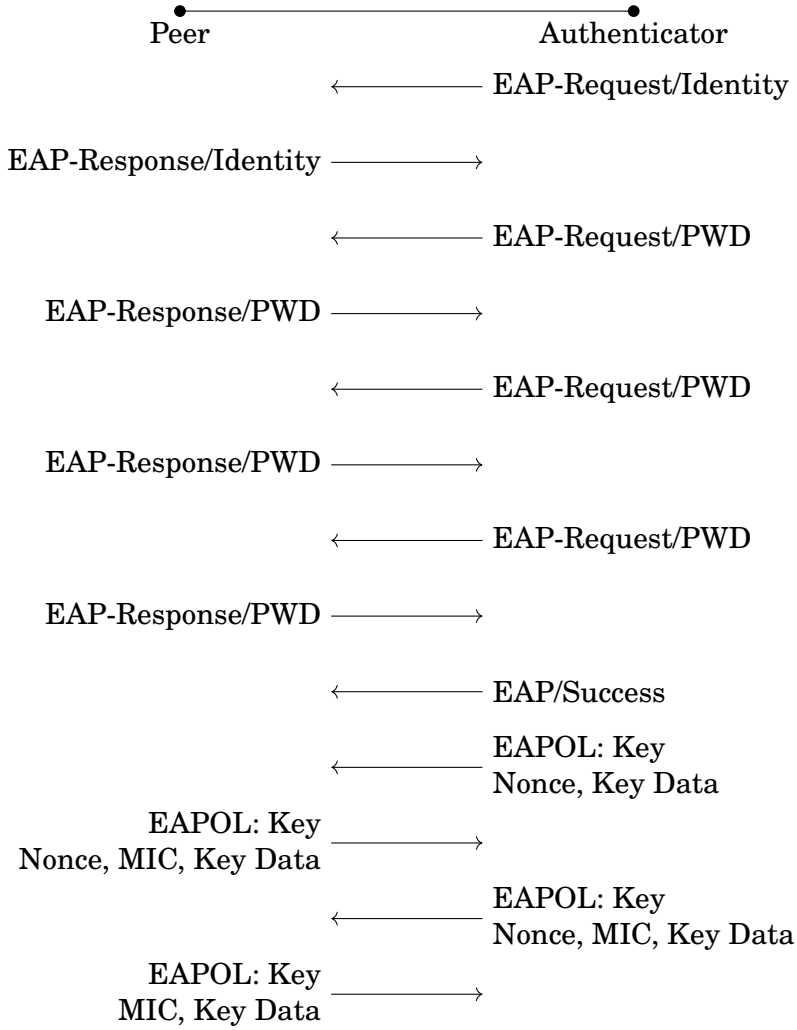

Figure 2.9. WPA2-Enterprise message exchange with EAP-PWD MIC: Message Integrity Check

3. Requirements for successful adoption

Our system design is driven by the following requirements:

- Our solution should not impact the WiFi access point, i.e., the EAP authenticator; only the authentication server and the client are updated. The number and the diversity of already deployed WiFi access points make their update an issue.
- The changes performed on EAP must ensure backward compatibility: a client or server not updated to support these changes must be able to successfully communicate with an updated end-point.
- The solution should not be restricted to a single EAP method; it should preferably fit with a large number of EAP methods, in order to be useful in many scenarios.
- The impact due to adding our solution to existing EAP method should be minimal to reduce the work needed for its adoption.
- The additional payload sent over EAP should be small; it must fit in a single EAP message to avoid fragmentation. (Recall that EAP does not include fragmentation.)
- The additional payload must be sent in a secure manner (ensuring its confidentiality, integrity and authenticity), because provisioning and configuration data is sensitive information.

4. System design: options for managing the client through EAP

As explained in the previous section, new solutions can be designed as extensions to EAP. Below, we identify several different options for adding information to EAP. The limits for each option are described, allowing a first selection on the most promising strategies to perform device management over EAP.

After describing the chosen architecture model, we specify what information must be added to EAP messages. The two last sections describe the possibilities for adding very small data and larger data in an EAP session.

4.1 Presentation of the architecture

In figure 4.1, communication A happens before communications B and C. The WiFi provider modifies the packets in this communication and transfers some packets to the authenticator. On the other hand, the WiFi provider does not modify packets in communications B and C. Communication A uses EAP since the peer does not have network access yet. After communication A, the peer obtains network access, and may obtain an IP address with the DHCP protocol. Then, communications B and C take place using another protocol, such as HTTPS.

The transition between communication A and communications B and C is ensured with additional data in the EAP session of communication A. Ideally, this additional data would be compatible with all EAP methods and not require fragmentation. It would only contain the necessary data to securely connect to the provisioning and configuration services in stages B and C.

In order to cover most possible situations, we separate the three server functions: authentication, provisioning and configuration servers. This results in an architecture that is adaptable to various types of networks. In figure 4.2, the peer contains three applications, each one communicating with a different server.

The authentication server verifies the identity of the client. The provisioning server manages

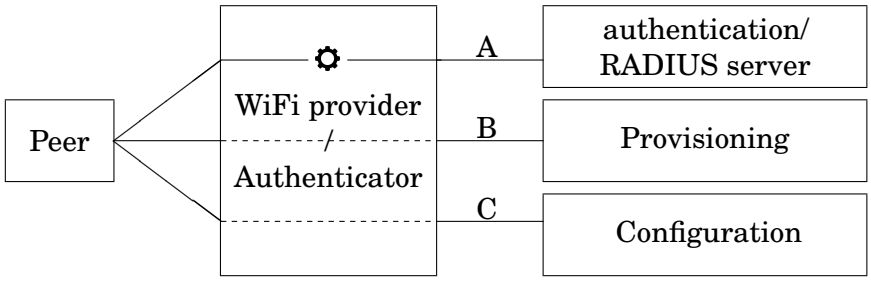

Figure 4.1. Entities in the management of a peer

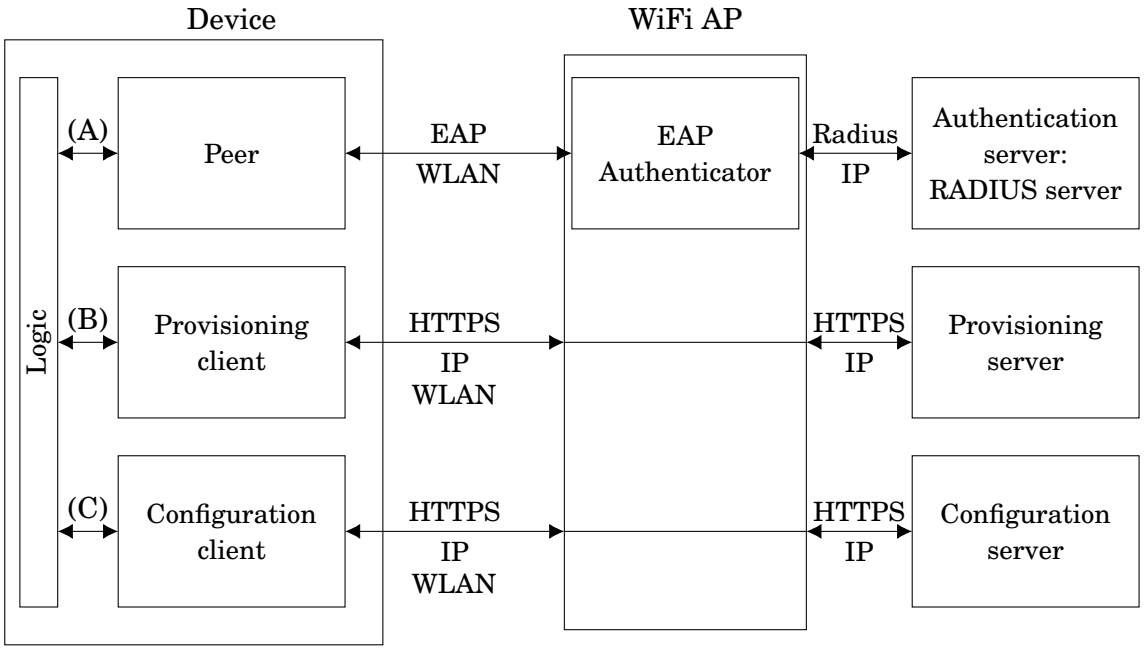

Figure 4.2. Description of the configuration - general presentation

the distribution of long-term credentials, e.g., certificates, to the client. The configuration server manages the configuration of the client. The three servers allow management of the client during its lifecycle. The role of each server is described in figure 4.2.

In (A), the client authenticates with EAP. The EAP method executed allows the authentication server to send a set of tokens for the device to perform provisioning and configuration.

In (B), using the information received in (A), the client connects to the provisioning server over HTTPS and obtains a certificate. This certificate allows reconnection to the network as well as strong authentication with other services on the network.

In (C), using the information received in (A), the client connects to the configuration server over HTTPS, authenticates to the configuration server using the certificate obtained in (B), and receives any information necessary for correct operation.

The transfer of the set of tokens between the client's applications can be performed either with a system call or with the storing of the data on the client's file system.

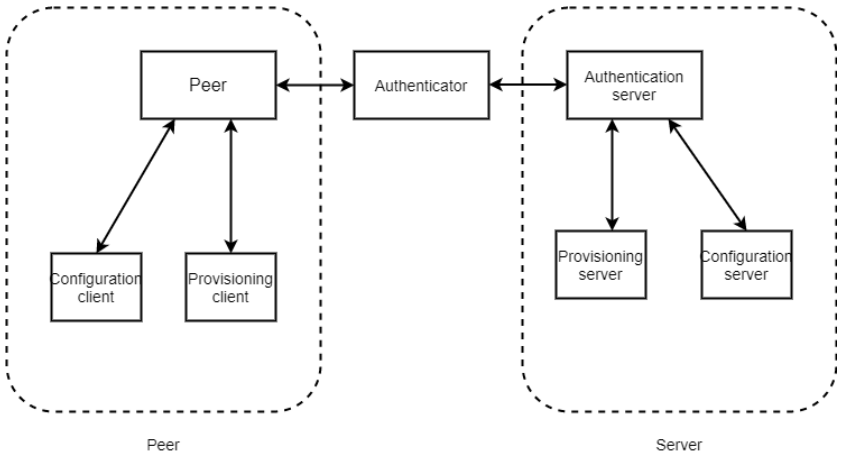

Figure 4.3. In EAP-CREDS, provisioning and configuration of the client is performed inside EAP session. EAP-CREDS is always run inside another EAP method, which provides security and fragmentation of the messages.

4.2 Advantages of this architecture

The management of a device at the beginning of its its lifecycle requires all three operations of authentication, provisioning and configuration. Authentication is necessarily the first operation to be executed. Because authentication is often performed with EAP, taking advantage of EAP to prepare the next steps – provisioning and configuration – can achieve easy adoption of new devices.

Another possible architecture is to perform all three steps within EAP. This is the goal of the draft EAP-CREDS [6]. However, EAP is initially adapted to the communication of short messages, whereas configuration may require arbitrarily large data transfer. It results in a less mutable setting: one server has to perform all three steps of authentication, provisioning and configuration. In addition, software development is faster in HTTP and REST services rather than inside the protocol stack.

The chosen architecture allows compatibility with more settings: the three servers may be in different locations, provisioning and configuration servers can use any protocol. The client may in addition perform regular configuration, whereas authentication is only required once: separating authentication and configuration allows easier configuration.

In comparison, the architecture proposed by EAP-TEAP and EAP-CREDS is more restrictive.

In both figures 4.3 and 4.4, all communication occur within EAP messages. In comparison, the strategy considered in this thesis operates with communications outside of EAP.

4.2.1 Content of the bootstrapping data

The bootstrapping data allows the client to securely connect to the provisioning and configuration servers. Therefore, the bootstrapping data must contain the following:

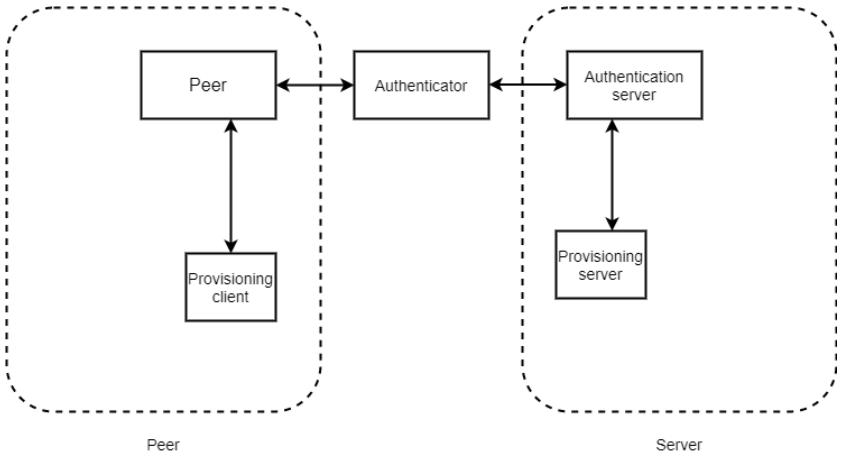

Figure 4.4. In EAP-TEAP, provisioning of the client is performed in EAP-TEAP through EAP messages. There is no standard configuration in EAP-TEAP.

- URL of the configuration server. The URL includes the protocol to use: for instance, configuration over HTTPS can result in the URL “https://config.com”.
- Credentials for secure access to the configuration server.
- Expiration date of the credentials to the configuration server.
- URL of the provisioning server. The URL includes the protocol to use: for instance, provisioning over EST can result in the URL “est://provision.com”.
- Credentials for secure access to the provisioning server.
- Expiration date of the credentials to the provisioning server.

4.2.2 Possible applications

The multiplication of device management protocols induces complexity: network managers must support multiple protocols at the same time, and private users must learn a new user experience for each device. Creating a unique protocol with preexisting standards allows standardization with a low cost of adaptation. Therefore, the aim is to modify one existing protocol as little as possible in order to allow long-term device management for all devices. EAP as initial protocol is justified by its widespread use and its inherent extensibility.

A device may have received from its manufacturer some initial credentials. However, these credentials cannot be used by the user as the device may change owner; hence each operational phase should use different credentials specific to the current user. Adding initial information for provisioning and configuration in the authentication step allows simplification of management: the device learns how to communicate with its environment in its first authentication. In addition, separating provisioning and authentication increases the adaptability of the protocol.

When adding new states between the factory and operational mode, it may become necessary to

isolate devices which have not yet performed provisioning on a specific network and give them limited access. This network separation is necessary only if additional security verification is performed with the provisioning and configuration steps. For instance, there may be device attestation [18] joined with provisioning: the device proves a set of claims that define its rights. Therefore, the device should execute the following steps upon initialization:

- Authentication: the device executes EAP authentication with either the manufacturer's credentials or an OOB channel. If successful, the access point adds the device to a restricted network.
- Provisioning and configuration: on the restricted network, the device can access the provisioning and configuration servers.
- Re-authentication: the device authenticates again with EAP, this time using the data obtained from the provisioning. Upon verifying the credential, the access point adds the device to the unrestricted network.

The distinction of the two networks is only necessary if the device has to perform additional identity validation during provisioning. Therefore, it is not necessary to separate the two networks in a home environment.

4.3 Secure communication transfer from authentication server to configuration and provisioning servers

4.3.1 Presentation

After the peer has successfully authenticated to the authentication server, it starts communicating with the configuration and provisioning servers. However, these two servers have not verified the identity of the peer yet. Therefore, they need to be ensured that the peer has successfully authenticated to the authentication server. This verification must happen before the peer receives configuration and provisioning data. This can be performed in several ways:

- OTP (One Time Password): the authentication server provides the peer with a password, and the peer then securely sends the password to the configuration and provisioning servers. This requires the servers to communicate securely. Alternatively, the OTPs can be generated sequentially from a master key shared between the three servers, hence avoiding additional communication. The sharing of a master key adds some requirements however: the servers need to be synchronized with the same time, and the key must have been pre-shared.
- Signature: the authentication server signs a message with the username of the peer and

a claim (stating that the peer has authenticated at some date). The peer can transfer this signed message as a proof of authentication to the configuration and provisioning servers.

4.3.2 One Time Passwords

In the successful concluding message sent by the authentication server to the peer, the server creates a unique password. Combined with the username, it will allow connection to the other two servers. The password is generated as:

```
Generate(counter, date, username)=HMAC(masterkey, counter);
counter++;
```

This ensures that no key is used twice. Then, the other servers can generate a list of possible passwords and verify that the one received from the peer is in this list. The limitation of this method is that the security relies on a single key, which has to be pre-shared among the servers. The sharing of the key can be more difficult if there are multiple instances of each server.

4.3.3 Signature

Transferring the authentication of the peer via a password is unnecessary: the authentication server can sign a proof that the authentication is successful. Therefore, the verification can be performed only with the public key of the authentication server - considering that the signature is performed with the authentication server's public key.

The message to be signed is as follows:

```
struct signed_message {
    timestamp end_validity;
    string username;
}
```

The signature algorithm used has to provide a short signature in order to save space on the EAP message. Therefore, RSA keys are avoided. Instead, the signature algorithm used is ES256 (SHA256 with ECDSA). The peer transfers this signed message to the next two servers, which can verify its validity with the public key they have previously stored. The presence of a datetime in the payload ensures that the peer does not use an old - and more easily leaked - authentication to obtain provisioning and configuration. This method is valid if the signed message is transferred securely from the authentication server to the peer. Therefore, the EAP method needs to encrypt the communication.

It is possible to put the signing time or the end of validity time in the signed message. However, choosing the end of validity time grants the delivering entity the choice for the validity duration of the token. Otherwise, the end of validity duration is chosen by the entity receiving the

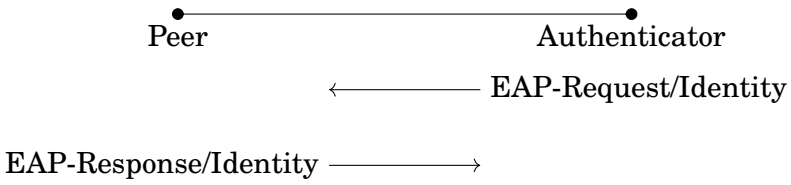

Figure 4.5. First message exchange in an EAP session

signed message. Giving the choice to the authentication server increases security since this entity is already strongly protected.

This second solution is selected for its ease of deployment. The signed message is implemented with JSON Web tokens. This format is defined in [19]. This format ensures ease of implementation and compatibility. The attribute end_validity is set to $now + 5\ minutes$ in order to avoid excessive use of the signed message.

4.4 Transfer of short information in EAP

In addition to the bootstrapping data, the EAP communication must include a negotiation on whether to send the bootstrapping data itself. Indeed, authentication is performed at every connection of the client to the network, whereas provisioning is necessary only on the first connection, and the configuration may follow a specific schedule of updates. There are several possibilities for adding short information to EAP conversation.

4.4.1 NAI or decorator

In Figure 4.5, the first message exchange in an EAP conversation is described. Each of the two messages include an anonymous username, i.e. a string with no restriction on its value.

EAP conversation starts with an Identity message on each side, including an anonymous identifier of the sender. The first EAP message is an EAP-Request/Identity from the authenticator, followed by an EAP-Response/Identity sent by the client to the authentication server. This identifier is not encrypted and therefore is not used in the authentication. It is possible to add a decorator to the user part of the NAI (Network Access Identifier) to share information on whether the server supports sending the bootstrapping data, and whether the client wants to receive the bootstrapping data. The decorator is appended to the user string. In some scenarios, the presence of a decorator can prevent correct authentication of either the server or the client. Therefore, background compatibility is not complete.

The first message EAP-Request/Identifier is sent from the authenticator, not the authentication server: if the authenticator can route to different authentication servers (e.g., the Eduroam network), the decorator in the first EAP-Request/Identifier from the authenticator must include whether each authentication server supports the sending of the bootstrapping data. Because

EAP messages have a limited size, it limits the number of authentication servers connected to the authenticator. This is necessary to inform the client whether the authentication server can send the bootstrapping data. As described in [20], the decorator is separated from the the username by the character for “end of string”.

The second message EAP-Response/Identifier is sent from the client. The client can add a decorator to specify whether it requests the bootstrapping data. However, if the authentication server does not support this update, the decorator might prevent correct authentication: the client must only add the decorator if the authentication server understands this decorator. If the client adds a decorator but the authentication server does not understand it, authentication will fail in spite of valid credentials.

This method has the advantage of being universal among all EAP methods: the Identity message exchange occurs before the initialization of an EAP method. The lack of backwards compatibility and diversity of application settings limits the use of NAI decorators, which goes against the requirements. The use of decorators has been described in [20]: although it confirms the possibility of adding NAIs in the first message exchange, standardization is not compatible with the large range of applications of EAP.

4.4.2 Client credential type in TLS handshake

Several common EAP methods use a TLS handshake in the first phase. The client can authenticate in the TLS handshake with a certificate. Alternatively, the client can authenticate after completion of the TLS handshake with a different type of credentials. The sharing of a certificate by the client in the TLS handshake can determine whether the client wants to receive the bootstrapping data or not. Indeed, if the client has already been provisioned with a certificate, it does not need to connect to the provisioning server again. The authentication server adapts its messages accordingly:

- The client sends a certificate: the server does not send the bootstrapping data.
- The client does not send a certificate: the server sends the bootstrapping data after authenticating the client with a different credential type.

This method is compatible with several commonly used EAP methods. However, it has two disadvantages. First, the specification of some EAP methods have to be changed in order to include sending of the bootstrapping data. Second, a client not updated to support this logic may have no certificate to provide: the reception of the bootstrapping data may lead the client to terminatie the EAP communication with failure.

4.5 Solutions for adding bootstrapping data

EAP communication strongly depends on the EAP method executed. In order to ensure compatibility among the EAP methods, we analyze how to add information to EAP messages. The bootstrapping data is a set of tokens allowing secure communications with the provisioning and configuration server.

4.5.1 Addition of a notification message

The EAP framework [1] defines fundamental types of messages: the type “Notification” allows the communication of a printable message. This is meant to transfer important information to the user that does not result in an error. Notification messages are by default allowed in EAP methods. The content of a notification message is not described in the specification, allowing its use for additional information. The communication of notification messages does not modify the state of the ongoing EAP method.

Extract from RFC 3748 [1]:

`The Notification Type is optionally used to convey a displayable message from the authenticator to the peer. An authenticator MAY send a Notification Request to the peer at any time when there is no outstanding Request, prior to completion of an EAP authentication method.`

Figure 4.6 describes the communication with an additional notification message containing the bootstrapping data in JWT format. This method is compatible with all EAP methods. It has two disadvantages:

- No security: the notification message is not encrypted or authenticity protected by default. Therefore, the content can be read by all on the network.
- Single message: the notification message does not allow two-directional communication: the client cannot request the bootstrapping data. Therefore, the notification message should be sent in every EAP execution.

Adding a notification message with a specific content requires modifying the behavior of the client and authentication server, and the EAP specification: the treatment of a notification message should differ depending on the content. This change in the EAP [21] specification is unlikely to be adopted.

4.5.2 Addition of an attribute to a preexisting message

Several EAP methods send attributes in the EAP payloads. EAP-TTLS uses AVP objects after the completion of the TLS handshake, while EAP-PEAP, EAP-TEAP, EAP-FAST, EAP-POTP

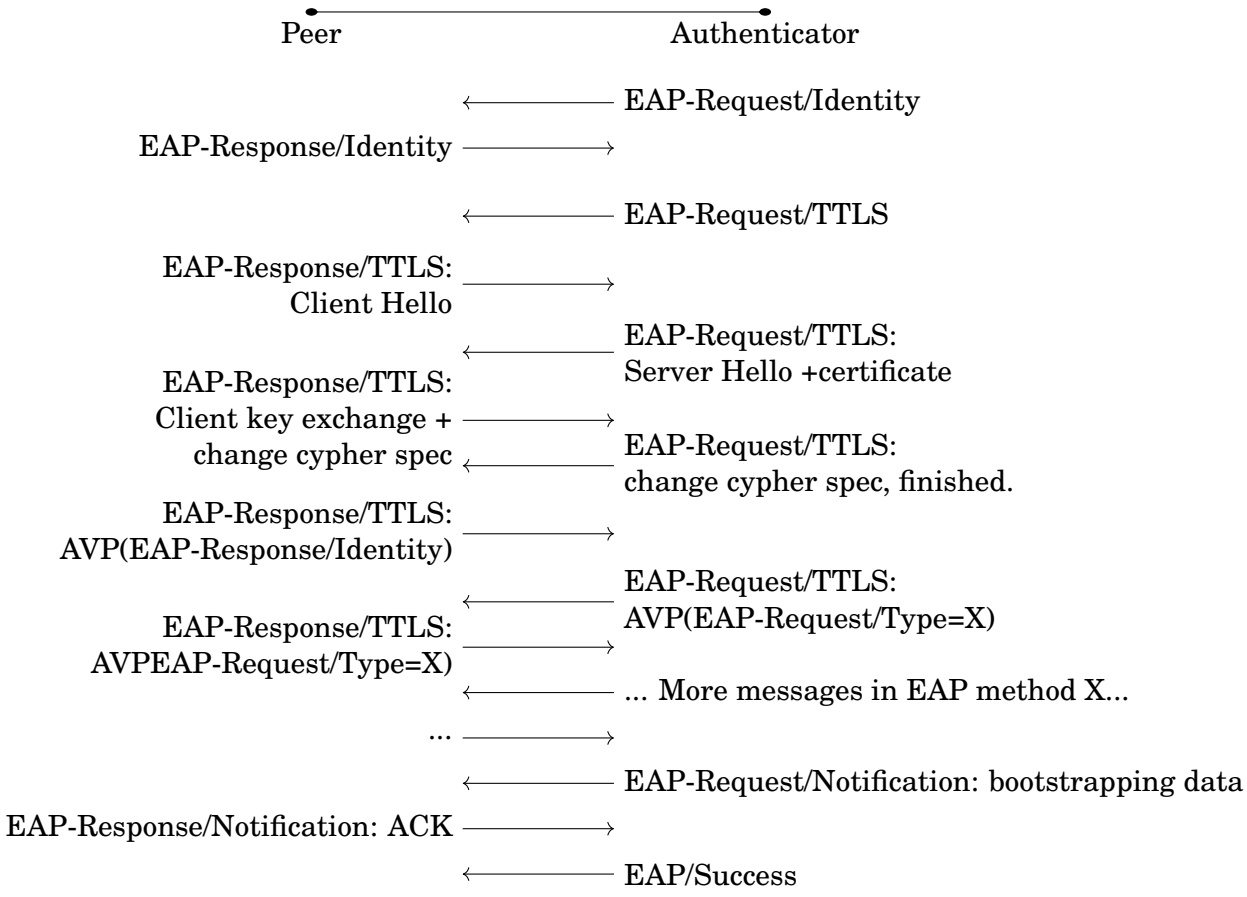

Figure 4.6. Messages in EAP-TTLS with a notification message carrying the configuration data

use TLV format. Adding an attribute with a new type requires modifying the selected EAP method specification. It avoids adding a new message, which would increase the duration of the communication. However, adding one attribute does not allow asking the client whether it wants the bootstrapping data. Therefore, the bootstrapping data has to be mandatory in the EAP conversation.

4.5.3 Addition of an inner EAP method

Some of the most widely used EAP methods set up a TLS tunnel for communication (e.g. EAP-TTLS, EAP-PEAP, EAP-TEAP, EAP-FAST). It is possible to execute another EAP method within the tunnel, hence taking advantage of the secure communication channel. The creation of a new EAP method, called EAP-iPROV (EAP-Inside PROVisioning), that occurs inside a TLS tunnel and sends the bootstrapping data is an efficient solution: it allows future improvements and respects the EAP specification. Figure 4.8 describes the protocol stack with EAP-iPROV. EAP-iPROV allows the bootstrapping data to be optional and only sent if the client requests it. It also avoids modifying previously existing specifications. Instead, only the configuration files of the client and authenticator need to be modified (supposing that both support EAP-iPROV).

Figure 4.9 describes the messages exchanged in EAP-iPROV when the peer wants to receive

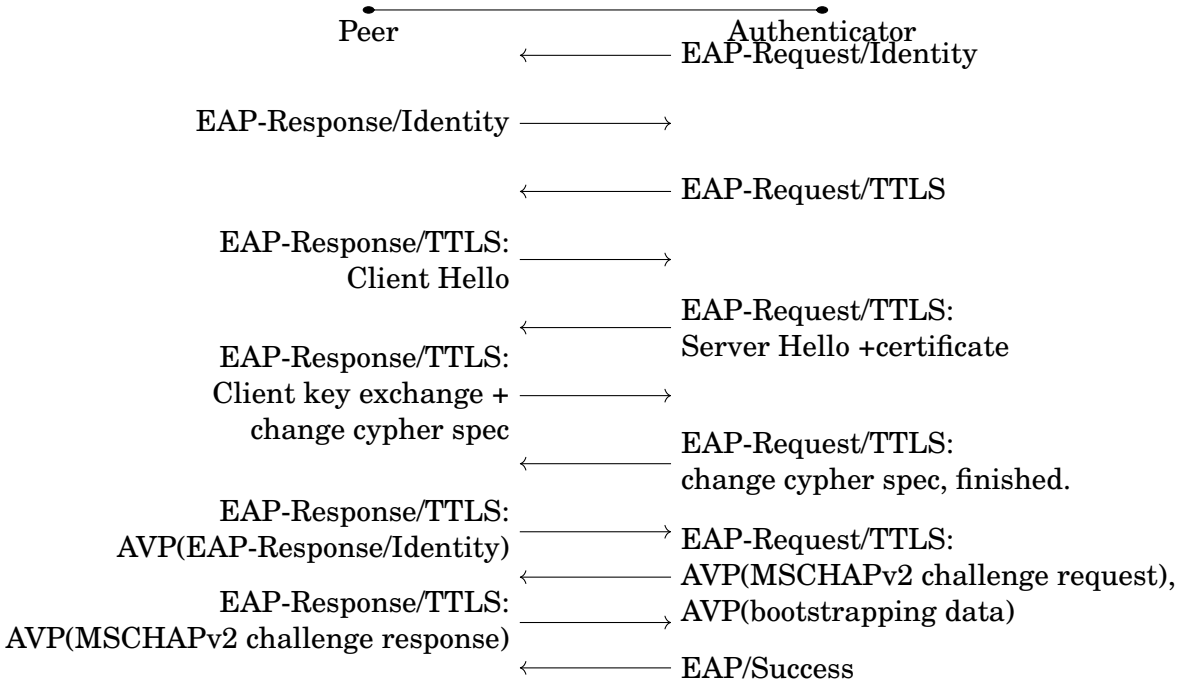

Figure 4.7. Additional attribute carrying the configuration data in EAP-TTLS message exchange

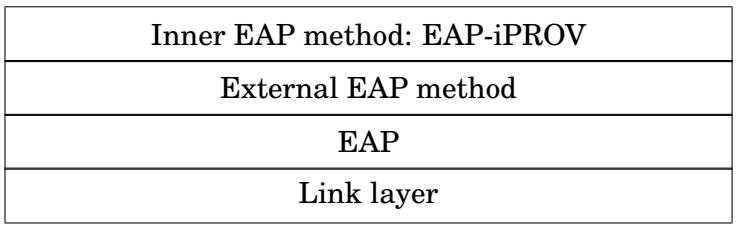

Figure 4.8. EAP-iPROV protocol stack

the bootstrapping data. If the peer is not interested, it returns in the first response a TLV version with value 0: the server stops EAP-iPROV, reaching an early success state. Therefore, EAP-iPROV only adds one message exchange when no token is necessary.

4.5.4 Addition of an outer EAP method

The creation of an additional EAP method, called EAP-oPROV (EAP Outside PROVisioning), allows encapsulating multiple successive EAP methods. Therefore, it is possible to execute inside EAP-oPROV the succession of one authentication EAP method, e.g., EAP-TTLS, and one configuration EAP method, e.g., EAP-iPROV, as described in Figure 4.10. This structure has the advantage of supporting all EAP methods, instead of only methods creating a tunneled communication. The security of EAP-iPROV is provided by encrypting the EAP-iPROV messages using the key materials from the authentication EAP method.

EAP-oPROV has one disadvantage: the bootstrapping data can be sent only if EAP-oPROV and EAP-iPROV are supported by the client. In comparison, adding an inner EAP method

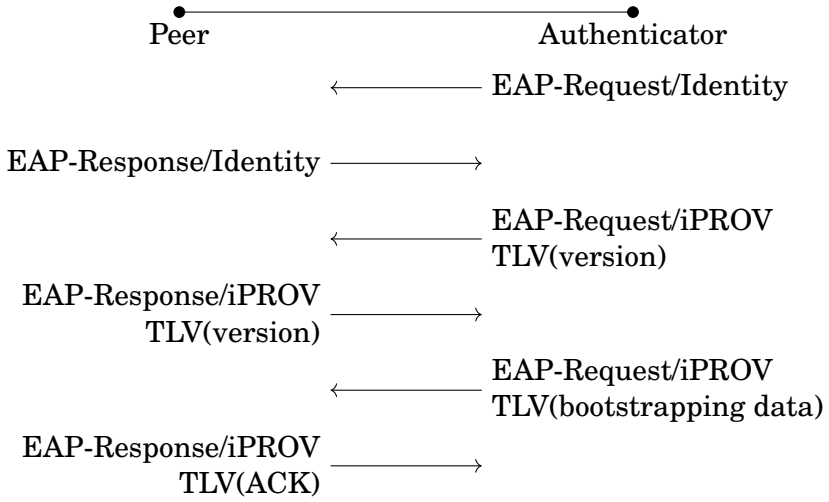

Figure 4.9. Messages in EAP-iPROV when the peer wants the configuration data

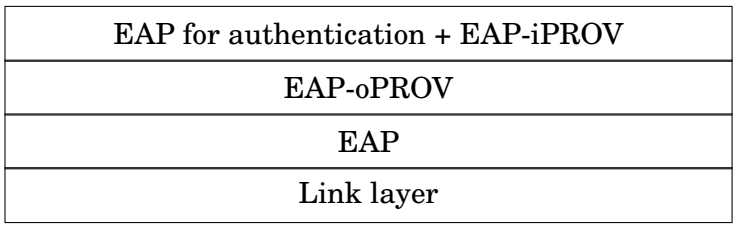

Figure 4.10. EAP-oPROV protocol stack

as described in 4.5.3 can be performed as long as the client supports EAP-iPROV and one standard EAP method that creates a master secret key (MSK).

4.5.5 Comparison of each solution among EAP methods

In Table 4.1, the strategies for adding the bootstrapping data are compared on each EAP method. The table is completed with no change allowed to the different methods' specifications. In the table, "+" indicates support; "-" indicates no support.

Adding a notification message requires changing the server's behavior for one EAP method, or for the EAP framework. Adding an attribute requires changing the specification of the method. An inner EAP method does not rely on specification change.

4.6 EAP-iPROV: design choices

The solution of adding an inner EAP method to transfer the bootstrapping data is implemented. This new method EAP-iPROV (EAP Provisioning Inside) has to follow requirements:

- No fragmentation is required. It allows maximum support on different external EAP methods.

EAP method	adding notification message	adding an attribute	allows inner EAP method	allows outer EAP method
EAP-TTLS [3]	+ (in tunnel)	+(AVP in tunnel)	+	+
EAP-PEAP [15]	+(by default)	-(see inner EAP method)	+	+
EAP-TEAP [2]	+(by default)	+(Vendor-Specific TLV, new TLV)	+	+
EAP-TLS [13]	+(by default)	-	-	+
EAP-FAST [4]	+(by default)	+(Vendor-Specific TLV, new TLV)	+	+
EAP-AKA [22]	+(by default)	+(new attribute type)	-	+
EAP-PWD [23]	+(by default)	-	-	+
EAP-POTP [24]	+(before last EAP message)	+(Vendor-Specific TLV, Protected TLV)	-	+
EAP-MAKE [25]	+(by default)	+(subtype values)	-	+
EAP-SIM [14]	+(by default)	+(new attribute type)	-	+
EAP-oPROV	+	-	+	+

Table 4.1. Comparison of available changes between EAP methods

- The server sends the bootstrapping data in a message. The data has to be authenticity protected.
- The client indicates if it wants the bootstrapping data.
- EAP-iPROV must verify the version of the method supported by each party.

The communication can be executed in two round trips. The different attributes are carried as TLV objects. Figure 4.11 describes the content of the messages when the client wants to receive the bootstrapping data. If the client does not want to receive the bootstrapping data, it sends the Version TLV with value 0: the server reacts to this value by terminating the method with success state.

The attribute EAP-Response Version TLV contains only one byte with the value of the version if the client wants to receive the bootstrapping data, and 0 otherwise.

When EAP-iPROV is executed as inner method, the identity message exchange at the beginning

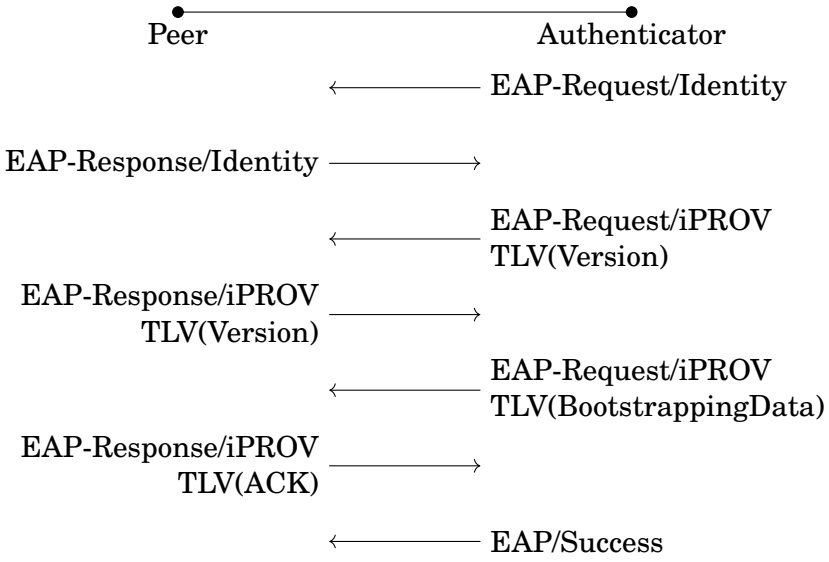

Figure 4.11. TLV objects included in the messages in EAP-iPROV.

and the Success (or Failure) message at the end are absent. EAP-iPROV requires the external EAP method to allow an inner EAP method to execute in an encrypted channel. This is possible with EAP-TTLS, EAP-TEAP, EAP-PEAP and EAP-FAST as external EAP method, as well as with EAP-oPROV, which will be discussed next.

4.7 EAP-oPROV: design choices

In order to increase the number of EAP methods that can be executed with EAP-iPROV, we can wrap the EAP communication in EAP-oPROV (EAP Provisioning Outside). EAP-oPROV carries TLV attributes. The following EAP types are defined:

- TLV-EAP: the content is an EAP message.
- TLV-ENCRYPTED: the content is an encrypted TLV object.
- TLV-VERSION: the content is the version of the protocol.
- TLV-ERROR: the content is an error type.
- TLV-SUCCESS: the content is a success message.
- TLV-FAILURE: the content is a failure message.

EAP-oPROV has two phases, described in figure 4.12:

- Phase one: authentication. Phase one performs an EAP method to authenticate the client, called the authentication method. Any EAP method that exports a key can be executed. At the end of phase one, if the authentication method is successful, EAP-oPROV takes the key material obtained from the authentication method. The key material can be used

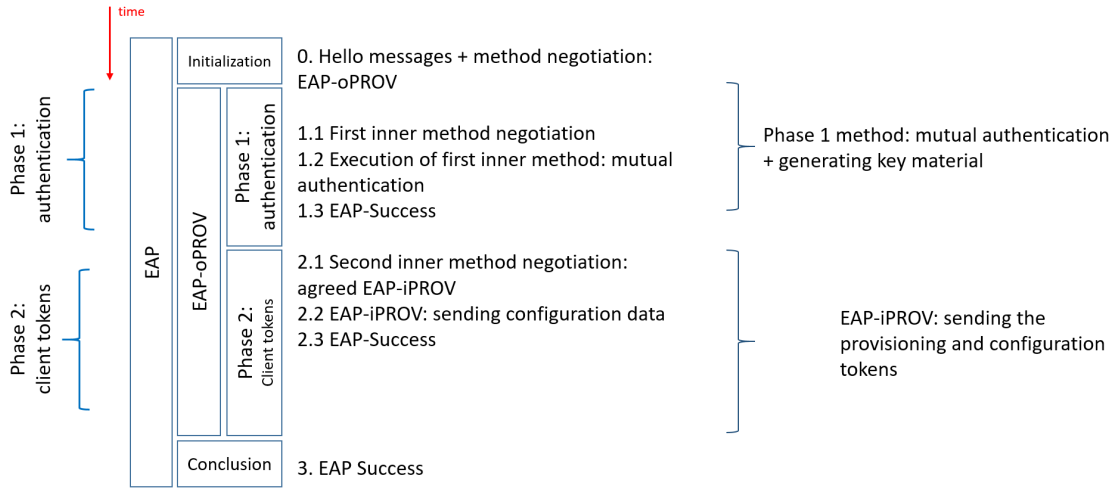

Figure 4.12. Structure of EAP-oPROV with EAP-iPROV in phase two

in phase two to encrypt TLV attributes.

- Phase two: configuration. Phase two can perform any other EAP methods, and each TLV object can be encrypted with authenticated encryption. EAP-iPROV can be performed with all messages encrypted. Once the server has executed all desired EAP methods, it reaches the state success.

Each inner EAP method allows a method negotiation between the peer and the server: this is allowed by starting a new EAP conversation for each inner method. This allows larger compatibility.

In order to securely support all inner EAP methods, EAP-oPROV must provide encryption and fragmentation.

4.8 Summary of the protocol options

Four options have been considered above. In Figure 4.13, these four options are summarized with their respective protocol stacks. EAP is supposed to run over the EAPOL protocol, although this is not a requirement.

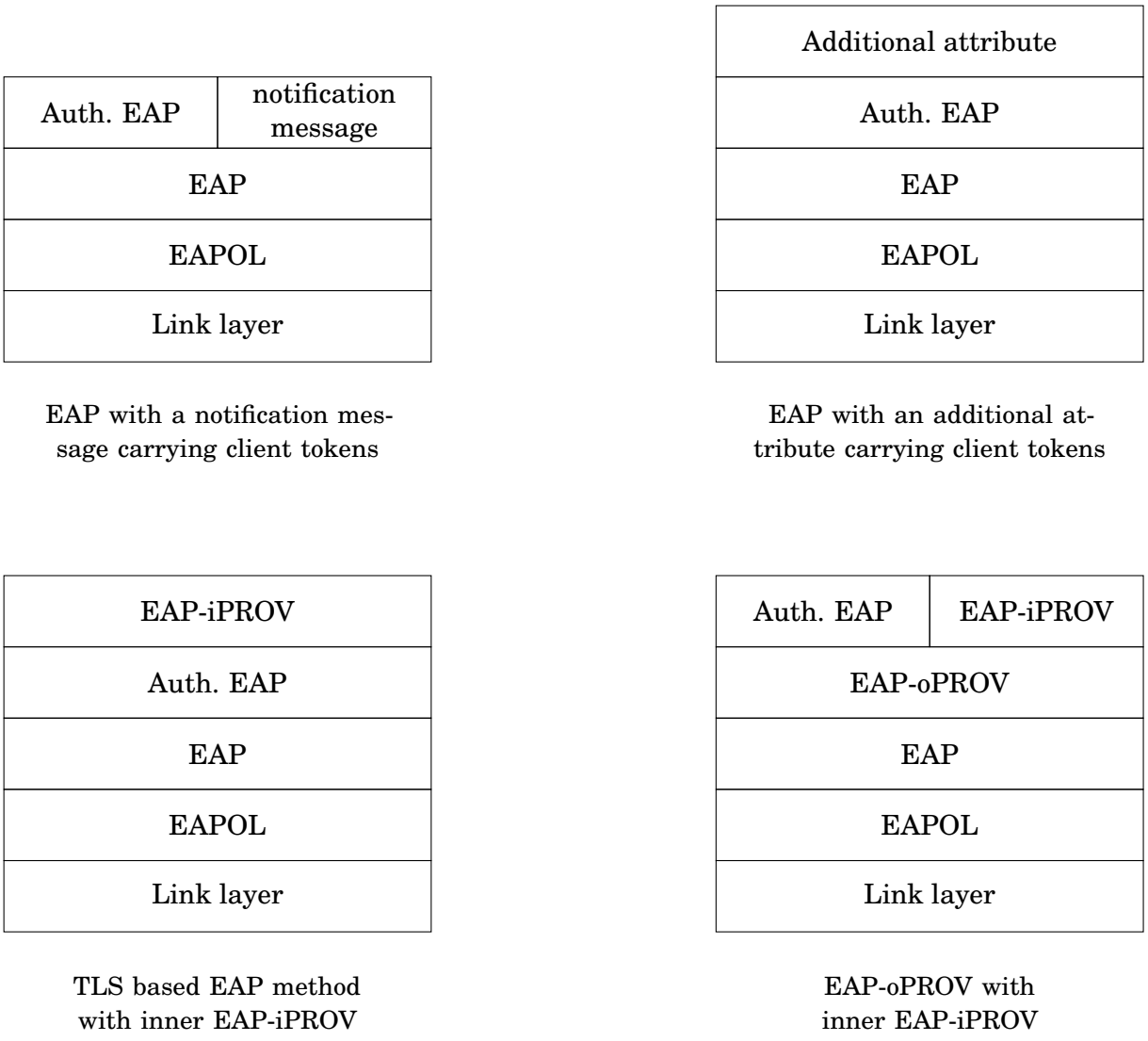

Figure 4.13. Protocol stacks of the four options for updating EAP. Auth. EAP is the standard EAP method used for authentication.

5. Implementation and performance comparison of different strategies

In order to accurately compare the solutions designed in the previous sections, each solution is implemented on the same testbed. After describing the implementation environment, we analyze the performance in order to determine which solution to retain. The comparison is based on the ease of integration of the solution in EAP, the number of bytes transferred in the EAP session and the duration of the EAP session. The reference for comparison is a widely used and secure EAP method, EAP-TTLS with inner MSCHAPv2 password verification.

5.1 General architecture

The proof-of-concept is implemented on two Raspberry Pi. The client is a Raspberry Pi 4 model B, connected to a router for ssh connection. The server is a Raspberry Pi 3 model B+, also connected to a router for SSH (Secure Shell) connection. The two Raspberry Pi communicate with each other over WiFi, as shown in Figure 5.1.

The client Raspberry Pi contains:

- WiFi client: this program manages the WiFi interface, and implements EAP.
- Configuration client: this program manages the configuration with the configuration server.
- Provisioning client: this program manages the provisioning with the provisioning server.

The server Raspberry Pi contains:

- WiFi AP (Access Point): this program creates a WiFi access point.
- Authentication server: this program authenticates the client with EAP. EAP messages are transferred by the WiFi AP.
- Configuration server: this program manages the configuration.
- Provisioning server: this program manages the provisioning.

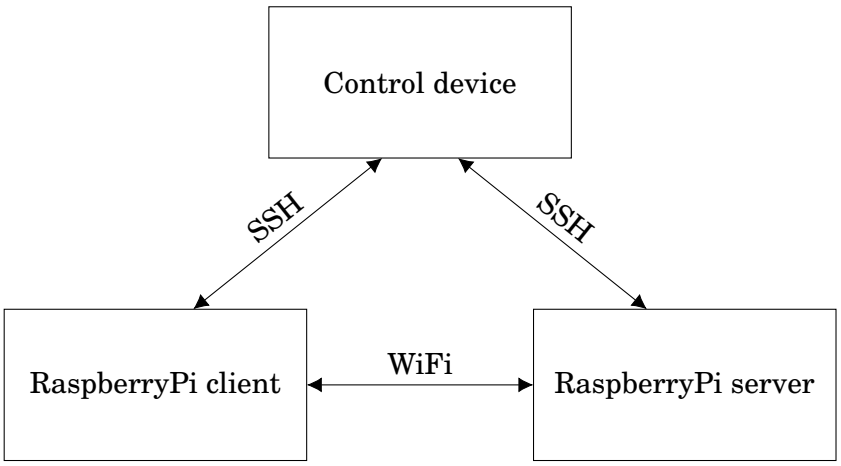

Figure 5.1. System experimental setup

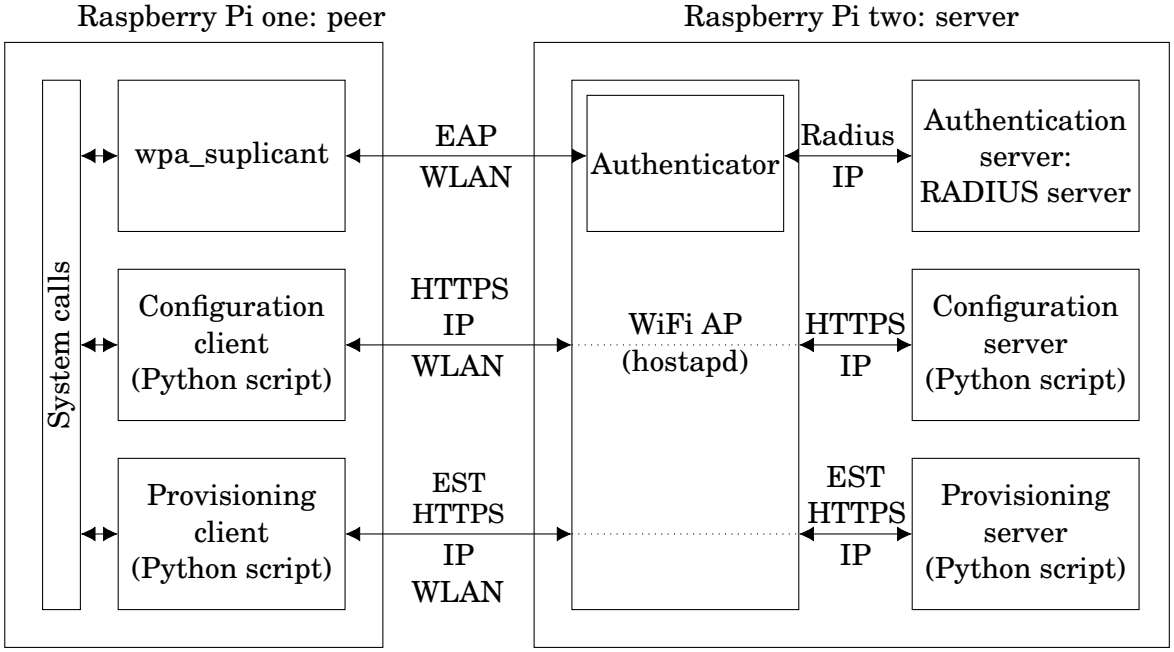

Figure 5.2. Description of the configuration, with implementation references - general presentation

The structure of the experimental setup is represented in Figure 5.2. The presence of both authenticator and authentication server in the same device is realistic, as the two programs operate on different ports.

5.1.1 WiFi client and authenticator

Both WiFi client and authenticator are implemented with the use of [26]. The authenticator is implemented with hostapd, and the WiFi client with wpa_supplicant. The configuration of the WiFi client is performed in the file */etc/wpa_supplicant/wpa_supplicant.conf*. It specifies the client's credentials and the EAP method to use. The WiFi AP is set to communicate with the authentication server using RADIUS protocol. The authentication server can verify the client's credential with the configuration files */etc/hostapd/hostad.eap_user* and */etc/hostapd/hostapd.radius_client*.

5.1.2 Provisioning and configuration components

The WiFi client (EAP peer) transfers the received tokens to the provisioning and configuration clients. We have implemented these in two Python scripts. Recall that the client tokens are JSON objects. The client uses these tokens to access the management servers over HTTPS.

The provisioning of the client certificate is done using EST protocol [27]. We used "Simple Python EST server + CA" [28] as the EST server. We have implemented the configuration server in a Python script.

5.1.3 Raspberry Pi

Raspberry Pi is a set of single-board computers developed by the Raspberry Pi Foundation. Initially meant for educational purposes as a low-cost computer, it is also used for other applications including robotics and proof-of-concept designs. The devices support several Linux based operating systems, including RaspiOS developed specifically for Raspberry Pis. Raspberry Pi has a strong community and many software libraries provide compatibility. These advantages determined the choice for Raspberry Pi as hardware in the proof-of-concept.

Raspberry Pi 3 model B uses a 1.2 GHz 64-bit quad-core ARM Cortex-A53 processor, with one gigabyte of RAM. Raspberry Pi 4 model B uses a 1.5 GHz 64-bit quad-core ARM Cortex-A72 processor, with four gigabytes of RAM. Both provide Ethernet connectivity with RJ-45 ports and WiFi access. Both Raspberry Pi use a 64 gB SD card, flashed with RaspiOS for operating system.

5.1.4 hostapd

Hostapd is a WiFi provider software. It is documented by the Raspberry Pi foundation as a solution for using a Raspberry Pi as a wireless access point. This insurance of compatibility and large support determined the choice of hostapd in the proof-of-concept. Jouni Malinen's implementation supports Linux and provides EAP authentication with use of a RADIUS (Remote Authentication Dial In User Service) server.

Several EAP methods are supported by Jouni Malinen's hostapd, including EAP-AKA, EAP-TTLS, EAP-TLS, EAP-SIM, EAP-PEAP, EAP-FAST, EAP-PWD, EAP-PSK and EAP-MD5.

It is possible to separate in two distinct processes the wireless access point from the RADIUS server. In this scenario, the two processes know each other's URL. The RADIUS server is configured with the client credentials from the file $/etc/hostapd/hostapd.eap\_user$. The RADIUS server can also use a database to store client credentials.

5.1.5 wpa_supplicant

wpa_supplicant is an open source software providing a WiFi client according to IEEE 802.11i[5]. It was chosen for its Linux compatibility, large EAP support and available implementation from Jouni Malinen. It supports WPA and WPA2 standards. wpa_supplicant includes an EAP client. It includes the parameters for client authentication: the credentials, allowed EAP methods and parameters of the methods.

5.2 Implementation of EAP modifications

Four strategies have been implemented:

1. Addition of a notification message.
2. Addition of an attribute in EAP-TTLS secure tunnel.
3. Addition of an inner EAP method, called EAP-iPROV, in EAP-TTLS secure tunnel.
4. Addition of an outer EAP method, called EAP-oPROV, supporting first EAP-TTLS, then a configuration EAP method.

5.2.1 Addition of a notification message

The EAP specification includes the message type “Notification”. It is sent by the authentication server to the peer and contains a human readable message. The message carries an important information but still optional. The client can ignore the Notification message.

The notification message is carrying an encrypted JSON object: it is not human readable. The client must correctly interpret this message, which requires a change to the EAP specification. The authentication server sends the notification message before the final success message, but after successful completion of the authentication algorithms. Therefore, it is only sent if the authentication is successful.

Although this solution is implemented, change to the EAP specification is rigorous and thus can take a long time before adoption.

Performance
We compare the performance of a normal EAP-TTLS/MSCHAPv2 session with an EAP-TTLS/MSCHAPv2 session including a notification message.

Table 5.1 underlines the additional payload for the server of 728 bytes. The client only has a small additional message to send. The notification message is received by the client without necessary replay by the server.

	messages from server	bytes from server	messages from client	bytes from client	duration (seconds)
EAP-TTLS/MSCHAPv2	8	1561	7	1987	0.313 [0.208, 0.419]
EAP-TTLS/MSCHAPv2 with notification message	9	2389	8	2005	0.504 [0.377, 0.620]

Table 5.1. Comparison of performance on EAP-TTLS with optional notification message
Each cell contains the median value, followed by the segment [minimum, maximum] values when the value may vary. The values are determined on a sample of at least 20 EAP conversations.

5.2.2 Addition of a notification attribute

This is implemented with EAP-TTLS method. Before reaching EAP-success state, the RADIUS server reaches an intermediate state in which it sends the notification message. This is achieved by modifying the RADIUS server. Once the client responds to this notification message, the server sends the EAP-success message. The notification message contains the tokens as a JSON encoded in UTF8.

Adding the notification as optional
On this conversation, we want to add the following information:

- Information one: the RADIUS server wants to inform the client that he supports the sending of a notification message
- Information two: the client wants to tell the RADIUS server whether he wants or not to receive the notification message

Information one can happen in several ways:

- In the message “EAP-Request/Identity” sent by the authenticator, there is an attribute containing a string, identifying the authenticator to the client (currently “hello”). It is possible to specify the string so that it informs the client whether the RADIUS server supports the notification message. However, this message is sent directly by the authenticator, not the RADIUS server. Therefore, the authenticator needs to know whether the RADIUS server supports the notification message beforehand. In addition, the authenticator may be configured with several authentication servers, each one managing a different client domain name: the authenticator must in that case inform the client with the list of authentication servers supporting the notification message. This increases the additional data to send in the message ’EAP-Request/Identity’.
- It is possible not to inform the client: the client simply tries to request a notification message and sees if the server reacts accordingly. If the client does not know information one, the behavior of the client must be compatible with the server independently of the

EAP version used (adding a modification to the username does not work for instance, because it may prevent correct authentication with not updated servers).

The second possibility satisfies the requirements and limits the implementation changes.

Information two can happen in several ways:

- Option A: the client can adopt a specific domain name. For instance, his identifier may be "ttls-user@iot.huawei.com" if he wants the notification message, and "ttls-user@huawei.com" if he does not want the notification message. This information is transferred in the message "EAP-Response/TTLS: AVP(EAP-Response/Identity), AVP(MSCHAPv2 challenge)" as the username. The disadvantage is that a server which does not support the notification message may be misled in incorrectly interpreting the username.
- Option B: the client can send a specific username in his first message "EAP-Request/Identity", which includes his anonymous username "ttls-user". The disadvantage is that a server which does not support the notification message may be misled in incorrectly interpreting the username.
- Option C: it is possible to take advantage of the different authentication methods of the client to obtain a different behavior of the RADIUS server. If the client authenticates with a password, the authentication server will consider that the client is authenticating for the first time, hence wants a notification message to perform a complete bootstrapping. If the client authenticates with a certificate, the authentication server considers that the client has been authenticated before, hence does not send the notification message.

Option C is the most promising: it is compatible with previous versions of EAP and does not require any additional change in the messages. Only the behavior of the RADIUS server has to be updated. However, in order to follow the TLS handshake protocol, the TLS options have to be specific. According to the RFC 5246 [16], the client authentication with a certificate occurs in the following condition:

- The server sends a CertificateRequest to the client.
- The client sends a list of certificates. The server must be able to validate the certificate authority of the chain. If the client does have any certificate to send, he must send a certificate list of length zero.
- If the server does not consider the client's certificate valid or if the client's certificate chain is of length zero, the document does not restrict the behavior of the server. Therefore, the server can either conclude the handshake with a failure message or continue the handshake without client authentication.

EAP-TTLS/MSCHAPv2	messages from server	bytes from server	messages from client	bytes from client	duration (seconds)
no provisioning	8	1561	7	1987	0.313 [0.208, 0.419]
provisioning with additional attribute	8 [8, 9]	2273 [2269, 3093]	7 [7, 8]	1987 [1987, 2011]	0.95 [0.835, 1.124]

Table 5.2. Comparison of performance on EAP-TTLS with optional configuration attribute
Each cell contains the median value, followed by the segment [minimum, maximum] values when the value may vary. The values are determined on a sample of at least 20 EAP conversations.

In the implementation, the sending of the tokens in a notification message is determined according to option C.

The information on whether the client successfully authenticated with a certificate is managed in the hostapd server by writing in a file. This is valid as proof-of-concept but does not support multiple client connections. Therefore, it is only for testing purposes. The TLS handshake is managed in hostapd by an external library (e.g., openssl): sending a variable from the external TLS library to the RADIUS program was not possible without changes to the TLS library. This problem was avoided by writing the variable in a file.

Performance
We compare the performance of one EAP-TTLS/MSCHAPv2 conversation with tokens in an optional attribute (with a client certificate), with a traditional EAP-TTLS/MSCHAPv2 (with a client certificate) conversation.

In Table 5.2, several differences appear:

- Duration difference: the addition of a notification message in the EAP session adds 0.2 seconds: the delay may come from the compiling and execution of the Python script generating the tokens.
- Message length difference: only the server sends more bytes with the optional configuration attribute. The increase is about 700 bytes.

5.2.3 EAP-iPROV in EAP-TTLS

Implementation choices
EAP-iPROV is implemented as a new EAP method. hostapd allows the definition of additional EAP methods. The creation of a new EAP method does not require any change in the authenticator. EAP-iPROV uses TLV objects with two bytes for type, two bytes for length, and the following TLV type codes:

- TLV(ConfigPayload): code 9.

	messages from server	bytes from server	messages from client	bytes from client	duration (seconds)
EAP-TTLS/MSCHAPv2	8	1561	7	1987	0.313 [0.208, 0.419]
EAP-TTLS/EAP-iPROV	10 [10, 11]	2400 [2392, 3177]	9 [9, 10]	2094 [2094, 2167]	0.943 [0.856, 1.094]

Table 5.3. Performance of EAP-iPROV in EAP-TTLS
Each cell contains the median value, followed by the segment [minimum, maximum] values when different values have been recorded. The values are determined on a sample of at least 20 EAP conversations.

- TLV(Version): code 8.
- TLV(ACK): code 10.
- TLV(Failure): code 11.

Although EAP-iPROV is kept as simple as possible in this thesis, it is possible to extend its applications by defining additional TLV types.

Performance
From Table 5.3, EAP-TTLS/EAP-iPROV results in a small increase in duration and number of bytes of the conversation compared with a standard EAP-TTLS/MSCHAPv2.

5.2.4 EAP-oPROV with EAP-TTLS

Implementation choices
EAP-oPROV follows the same format of TLV objects as EAP-iPROV: two bytes for type, two bytes for length. The TLV types are different:

- TLV(Encrypted): code 2.
- TLV(Version): code 8.
- TLV(EAP): code 7.
- TLV(Success): code 10.
- TLV(Failure): code 11.

Encryption
The encryption is performed on one TLV object, including the type and length bytes. It uses the AES-GCM algorithm for authenticated encryption. The key is derived from the 64 bytes long *Master Session Key* (MSK) of the key materials obtained in phase one from the inner authentication EAP method.

However, obtaining the key material from the inner phase one method may depend on the

	messages from server	bytes from server	messages from client	bytes from client	duration (seconds)
EAP-TTLS/MSCHAPv2	8	1561	7	1987	0.313 [0.208, 0.419]
EAP-oPROV/ EAP-TTLS+EAP-iPROV	11 [11, 12]	2432 [2416, 3179]	10 [10, 11]	2150 [2150, 2193]	0.981 [0.879, 1.148]

Table 5.4. Performance of EAP-oPROV with two inner methods: EAP-TTLS followed by EAP-iPROV. The values for each method are based on a sample of about 100 EAP sessions. Each cell contains the median value, followed by the [minimum, maximum] range.

software used. Indeed, the key materials generated by an EAP method is available after the EAP method has reached the status "SUCCESS". This status is reached upon successful conclusion of the method. The state SUCCESS is reached differently by the client and the server:

- Server: since the server sends the Request messages, it determines which message in the method will be the last. Upon sending of this last message, the server switches to state SUCCESS.
- Client: the client determines the EAP method as successful upon reception of a message with content only sent as final step of the method.

Therefore, the key materials are available once the last message of the EAP method is received by the client. This implies that there is no need for an EAP-Success message in order to access the key materials. In EAP-oPROV, the absence of EAP-Success message on conclusion of an inner method is not problematic to access the key materials generated by the inner method.

Performance

In Table 5.4, EAP-oPROV is compared with a standard EAP-TTLS method. The addition of client tokens adds around one kilobyte of data sent from the server. The duration increases by 0.7 seconds: this is acceptable for registering a device.

Example application

In Figure 5.3, EAP-oPROV encapsulates EAP-TTLS for authentication, and EAP-iPROV for configuration. EAP-iPROV can be encrypted with the key materials from EAP-TTLS, although this is not shown in this example. In the implementation, there is an inner EAP-Identity exchange before starting any inner EAP method. This is unnecessary in practice.

5.3 Separation of networks

In order to secure the devices at the beginning of their lifecycle, it may be required to separate devices in IA (initial authentication mode) 2.2 state from devices in O (operating mode) state. For instance, devices in IA state may only have access to the provisioning and configuration servers, whereas devices in state O may have unrestricted access. This separation is implemented in the WiFi access point and can be done on different protocol layers. For example,

security can be implemented in the application layer 5.4 by requiring the client to provide a certificate for use of application on the local network. This section focuses on two possible solutions in order to separate networks.

5.3.1 Separation with VLANs

The access point may assign the client to a specific VLAN, depending on its state. The assignment to one VLAN is determined by the authentication server.

First, the device performs initial authentication with the manufacturer's credentials. The authentication server concludes this EAP conversation with a success message to the access point, joined with an assignment of the device to VLAN 1. On this VLAN, the client only has limited access to the network. For instance, the client may only access the configuration and provisioning servers. Once the client is provisioned and configured, it authenticates to the access point with the credentials obtained during provisioning. The authentication server assigns the client to VLAN 2 upon success of the authentication. Once the access point assigns the client to VLAN 2, the client has full access to the network.

This operation is possible with specific software. The authentication server must support dynamic VLAN assignment. This is only possible with specific RADIUS server implementations – for instance FreeRadius. The implementation of the RADIUS server provided by Jouni Malinen's implementation does not support dynamic VLAN assignment. The access point must accordingly support dynamic VLAN assignment.

5.3.2 Separation with SSIDs

The access point may advertise two networks with different access rights. The client identifies to the access point on network one using its manufacturer's credentials. It then performs configuration and provisioning. Afterwards, the client can identify to the access point on network two using the credentials obtained during provisioning. The process is similar to the use of VLANs.

This operation is possible with specific hardware. The access point must be able to manage two networks: this requires a specific WiFi card. The ability of a WiFi card to provide several SSIDs can be verified by the command "iw list" on Linux based systems.

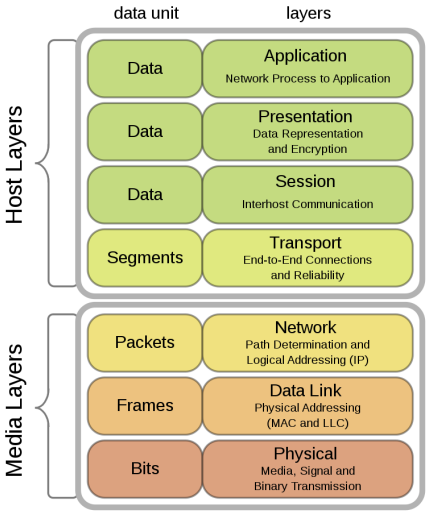

Figure 5.4. Protocol stack
Source: Wikipedia

```
> iw list
    ...
    valid interface combinations:
```

```
        * #{ managed } <= 1, #{ P2P-device } <= 1, #{ P2P-client, P2P-GO } <= 1,
             total <= 3, #channels <= 2
        * #{ managed } <= 1, #{ AP } <= 1, #{ P2P-client } <= 1,
             #{ P2P-device } <= 1, total <= 4, #channels <= 1
    ...
```

The attribute “#{ AP } <= 1” determines the maximum number of SSIDs supported by the access point. The WiFi card present on a Raspberry Pi only supports one SSID. Therefore, the implementation relies on the addition of a WiFi card connected to the Raspberry Pi hosting the access point.

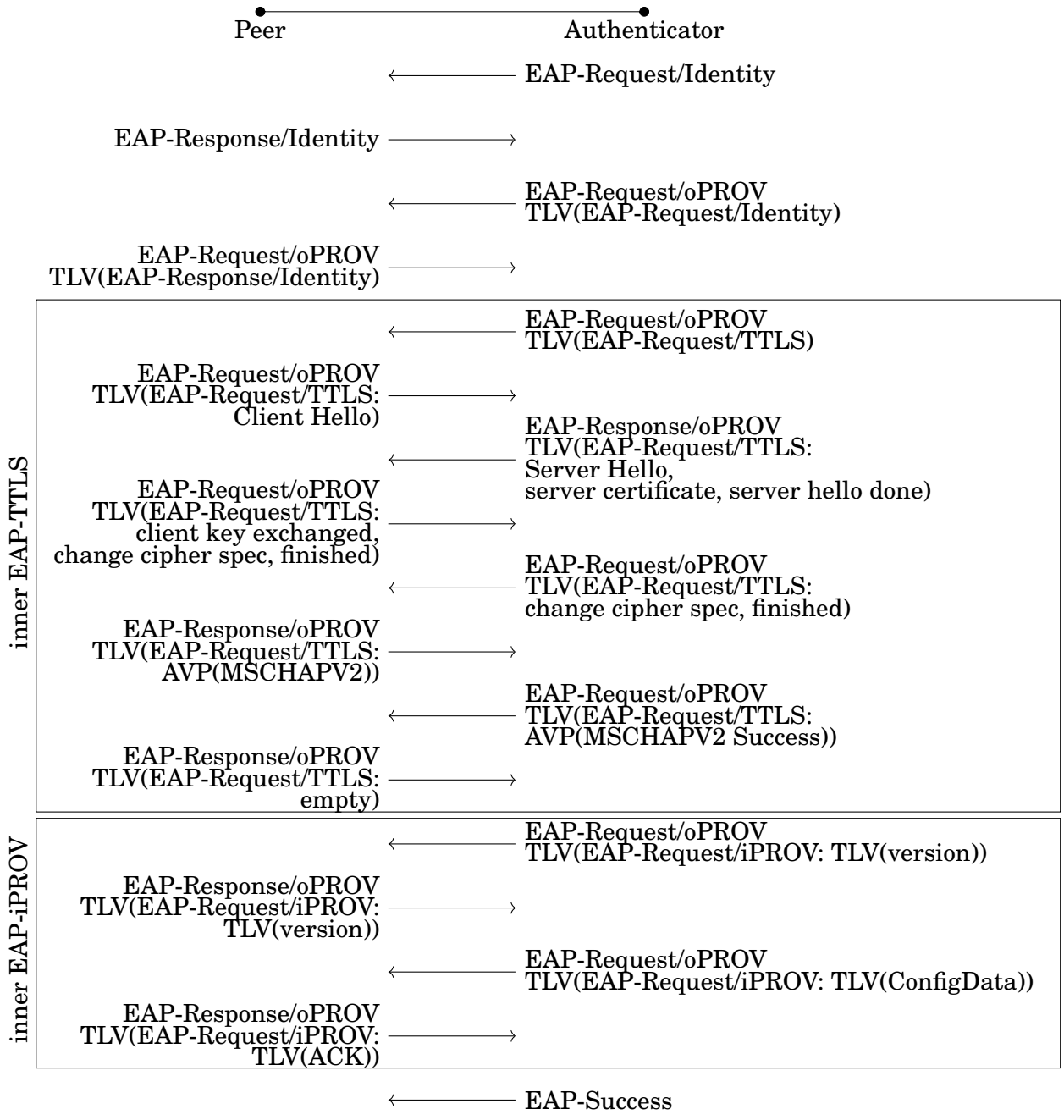

Figure 5.3. Messages in EAP-oPROV with inner EAP-TTLS and EAP-iPROV, when the peer wants the configuration data

6. Analysis and discussion

In the previous chapters we have outlined several protocol options for initiating provisioning and configuration of the client using EAP:

i. Sending the provisioning data to the client in EAP notification message.

ii. Adding a provisioning attribute to a some existing EAP method's message.

iii. Running an EAP-iPROV method without EAP-oPROV inside a TLS tunnel configured by an outer EAP method;

iv. Using EAP-oPROV as a wrapper for multiple EAP methods.

There are issues with the protocol option (i). First, the client cannot request the configuration data or reply to it. Second, a notification message is specified in [1] to contain a printable message destined to the user. This is different from our intended use.

There are issues also with protocol option (ii). The first is the same as above: the client cannot request the configuration data or reply to it. The second issue is that adding an attribute in a pre-existing message is specific to a single EAP method. If we want to add provisioning and configuration in another EAP method, then a similar modification has to be applied.

The applicability of (iii), running an EAP-iPROV method inside a TLS tunnel configured by an outer EAP method, without EAP-oPROV, is limited to those EAP methods that use a TLS tunnel.

Using EAP-oPROV in (iv) as a wrapper for multiple EAP methods has the following advantages. First, since a large set of EAP methods can run inside EAP-oPROV, there is a large set of different authentication methods that can be used with EAP-oPROV. Second, EAP-oPROV is flexible with respect to the choice of inner methods. In addition to EAP-iPROV one or more inner EAP methods can be run inside EAP-oPROV; moreover, EAP-iPROV can be replaced by another inner method. For instance, EAP-oPROV could execute the draft EAP-CREDS [6] after the first inner method.

In summary, EAP-oPROV is the most promising protocol option to initiate provisioning and configuration with EAP.

In the rest of this chapter we will analyze the security of EAP-oPROV, and then compare it with the approach where provisioning and configuration of the client are done entirely inside EAP-TEAP. We then describe a series of experiments to evaluate the latency and communication overhead of different protocol options. The chapter ends with section about future work.

6.1 Security analysis of EAP-oPROV

Recall that EAP-oPROV operates in two phases (see Figure 4.12). In phase one, a generic EAP method is executed for authentication. This first inner EAP method provides authentication and key establishment. In phase two, a second inner EAP method is executed to share a configuration message. This second inner EAP method is encrypted using the key materials from the first inner EAP method.

The following requirements are present for a secure EAP-oPROV:

- In the authentication server:
  - Only allow secure EAP methods with mutual authentication, e.g., EAP-TTSL, EAP-TLS, EAP-FAST, EAP-PEAP, EAP-AKA, EAP-SIM, EAP-AKA', EAP-NOOB.
  - Authentication methods must be up-to-date, e.g., EAP-TTLS v1 [29], EAP-PEAP v2 [30].
- In the client:
  - Only allow secure EAP methods with mutual authentication, e.g., EAP-TTSL, EAP-TLS, EAP-FAST, EAP-PEAP, EAP-AKA, EAP-SIM, EAP-AKA', EAP-NOOB.
  - Authentication methods must be up-to-date, e.g., EAP-TTLS v1 [29], EAP-PEAP v2 [30].
  - The JSON Web Tokens must only be sent over an encrypted and authenticated channel.
  - The certificate of the authentication server must be verified.
- In the authenticator:
  - The connection to the authentication server must be secure, and the authenticator must not be corrupted.

6.1.1 Securing second inner EAP method

Apart from EAP-MD5, all standard EAP methods generate a *master session key* (MSK) upon successful completion. The MSK is at least 64 bytes [1] and can be used to encrypt further communications. The generation mechanism of the MSK depends on the EAP method executed. The security of the MSK varies accordingly.

The confidentiality of the second inner EAP method for configuration is ensured by the properties of the phase one EAP method. This increases the importance in the choice of the phase one EAP method. The phase one EAP method should provide mutual authentication in order to avoid man-in-the-middle attack. In a mutual authentication method, both ends must respond to a challenge on the password in order to prove their identity. These challenges avoid corruption of the messages by an intermediate attacker. In addition, the master session key generated by the phase one EAP method must depend on all the authentication algorithms used during phase one.

In that regard, several EAP methods should be avoided. For instance, EAP-TTLSv0 generates a master session key from the TLS handshake parameters. Therefore, the inner authentication method – for instance MSCHAPv2 algorithm – does not influence the resulting MSK. As a result, it is possible for an attacker to act as a man-in-the-middle and obtain the correct MSK, without knowing the pre-shared secret.

This problem occurs with EAP-TTLSv0 [3] and is solved in EAP-TTLSv1 [29] by binding the MSK to the inner authentication method. EAP-PEAPv1 [15] is protected against man-in-the-middle attack by using the compound authentication binding [31] method, and EAP-PEAPv2 [30] binds the MSK to the inner authentication method. Compound authentication binding adds a message exchange containing MACs of all MSKs generated during the EAP session. However, hostapd implementation implements version 1 of EAP-PEAP without compound authentication binding: the MSK only depends on the parameters of the TLS handshake. Therefore, this implementation is insecure against the man-in-the-middle attack.

The MSK is used to create a symmetric key:

$$key = hmac\_sha256\_kdf(MSK, label = \text{“Derive EAP-iPROV message key”}||\\ \text{phase 1 method name}||\text{EAP-oPROV identifier}, salt = \text{peer identifier}, length = 16)$$

The symmetric key *key* is then used for encryption of the messages with the AES-GCM algorithm:

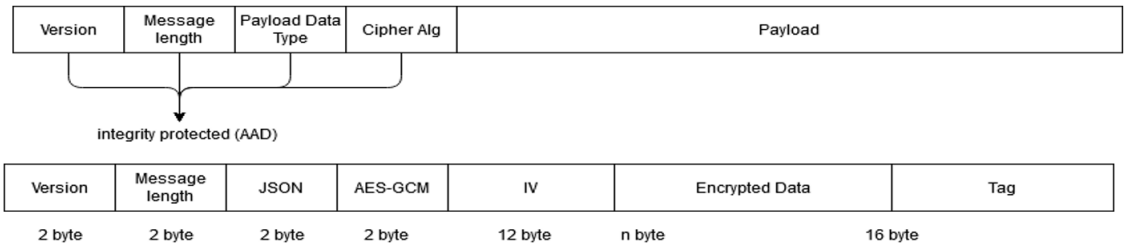

Figure 6.1. Encrypted message format in EAP-oPROV

The first message description focuses the content of the AAD (associated additional data). The second message description focuses on the payload structure.

$$(enc, tag) = aes\_gcm(message, AAD, key = key, iv = \text{12 random bytes})$$

The AAD is the associated additional data: it provides information on the encryption algorithm, version and data type. The message sent contains the elements enc, aad, iv and tag. The encrypted message format is described in Figure 6.1.

The key includes identifiers specific to the phase one method with the phase one method name, peer identifier and EAP-oPROV identifier. These elements ensure that the key is different in each EAP session. AES-GCM [32] provides authenticated encryption with associated data (AEAD).

6.1.2 Threats and attacks

Several attacks are possible during the execution of EAP. First, if the peer or the authentication server is compromised, the attacker has access to the credentials stored in the device: authentication becomes unsecure. Therefore, we consider situations where the peer and the authentication server are secure. The attacker can either manage to corrupt the access point, or act as man-in-the-middle. In either case, the attacker has access to all EAP messages and can modify them. In a man-in-the-middle attack, the attacker acts between the peer and the authenticator: it impersonates the authenticator to the peer, and the peer to the authenticator. In practice, the attacker provides a WiFi access point to which the client can connect. In this scenario, the attacker must be unable to successfully modify and replay the messages. If some part of the EAP session is encrypted, the attacker must be unable to decrypt it.

In order to avoid replay attacks, the session must include a nonce: in EAP-oPROV, the nonce is created in the phase one inner method – which is secure according to requirements. This nonce must be transferred to phase two: the encryption is performed using the MSK generated

$\log_2(N)$	$p(k,N)$				
	$5 \cdot 10^{-1}$	$1 \cdot 10^{-1}$	10^{-3}	10^{-6}	10^{-9}
32	$1.5 \cdot 10^{5}$	$6 \cdot 10^{4}$	$6 \cdot 10^{3}$	$1.86 \cdot 10^{2}$	$7 \cdot 10^{0}$
64	10^{10}	$4 \cdot 10^{9}$	$4 \cdot 10^{8}$	$12 \cdot 10^{6}$	$4 \cdot 10^{4}$
128	$4 \cdot 10^{19}$	$1.7 \cdot 10^{19}$	$1.65 \cdot 10^{18}$	$5.2 \cdot 10^{16}$	$1.6 \cdot 10^{15}$
256	$8 \cdot 10^{38}$	$3.1 \cdot 10^{38}$	$3 \cdot 10^{37}$	$9.6 \cdot 10^{35}$	$3 \cdot 10^{34}$

Table 6.1. Number of hashes to be computed to reach a given probability of collision $p(k, N)$, depending on the length of the hash.

by phase one. The MSK includes a nonce according to the requirements – security of the phase one method. For instance, EAP-TTLS uses TLS handshake parameters to create the MSK, a nonce being included in these parameters. Therefore, replay attack is only possible in EAP-oPROV if it is possible in the phase one inner method.

The security of the encryption in phase two has been discussed in Section 6.1.1.

The ability to modify messages in phase one depends on the security of the phase one method used. The requirements listed in 6.1 restrict accordingly the EAP methods to be used.

6.1.3 Security of JSON Web Tokens

The configuration data is structured with JSON Web Tokens. The algorithm used for signing is ES256, which is ECDSA using P-256 and SHA-256. This algorithm uses an asymmetric key pair, which allows easy verification by multiple servers, while only the authentication server can generate the token.

Within the tokens, the authentication provides a truncated hash of a certificate for each server the client should trust. The truncated hashes are chosen of 16 bytes long. An attack is possible by finding a collision attack. This supposes a choice in the certificates of each server: this attack can only be performed by the manufacturer itself.

The probability $p(k, N)$ of collision of k hashes is

$$p(k,N) = 1 - \frac{N!}{(N-k)!N^k} \approx 1 - e^{-\frac{k(k-1)}{2N}}, \tag{6.1}$$

where N is the number of possible values of the hashes, and $k \ll N$ [33]. Table 6.1 summarizes the values of k that would result in a given probability of collision $p(k, N)$ for $N = 2^{32}, 2^{64}, 2^{128}$ and 2^{256}.

The choice of 128 bits for hash length is sufficient to obtain protection against collision attack.

The use of a JSON Web Token without a password supposes that the token will never be shared with an attacker. In the JSON object sent from the authentication server to the client, the client token is joined with the server's certificate hash which should receive the token. When

the client connects to the provisioning or configuration server, it should receive and verify the server's certificate before sending the client token protected by encryption. In addition, the expiration date in the client token limits the leaking of a token: an attacker having obtained the token can only use it for a limited time.

6.1.4 Peer anonymity

In an EAP session, the peer sends its identity to the authenticator in its first message, before starting any EAP method. The username sent in this first message allows the authenticator to decide which authentication server to connect to based on the domain name appended to the username.

EAP-oPROV can support *peer anonymity*, where the peer sends an anonymous identity, e.g., anonymous@aalto.fi, to the authentication server in the first EAP-oPROV message, and the actual identity, e.g., sebastien.boire@aalto.fi, is sent afterwards encrypted in the first inner method. This would protect user privacy from eavesdroppers on the network path between the client and the authentication server.

One example of first inner method that conceals the actual identity is EAP-TTLS, when it is used in such a way that the client identity is sent inside encrypted TLS channel. Another example is EAP-TLS when it is used with TLS 1.3 in such a way that only the server is authenticated during handshake; after the TLS channel is established, the client is authenticated using Post-Handshake Client Authentication mechanism of TLS 1.3.

6.2 Comparison with EAP-TEAP

EAP-TEAP includes provisioning of certificate to the peer within the TLS tunnel established by the method ([2], section 3.8.2). EAP-TEAP does not have a mechanism for sending configuration data to the peer, but it has a possibility to add vendor-specific extensions to the protocol. Thus, it is possible to add configuration data within this method by using Vendor-Specific TLV inside the TLS channel, and we have implemented this extension in EAP-TEAP.

Please note that in order for this addition to work, both peer and authentication server have to understand how to interpret this TLV, and vendor-specific TLVs are not standardized. This restricts the applicability of adding client configuration to EAP-TEAP in vendor-specific extension.

Configuration data can be large, for example if it contains a device's firmware update. After implementing this extension, we have found experimentally that EAP peer exits with failure when the configuration data is large. The reason is that in hostapd implementation of EAP-TEAP the maximum size of any message is at most 16 kB, i.e. the configuration data cannot be

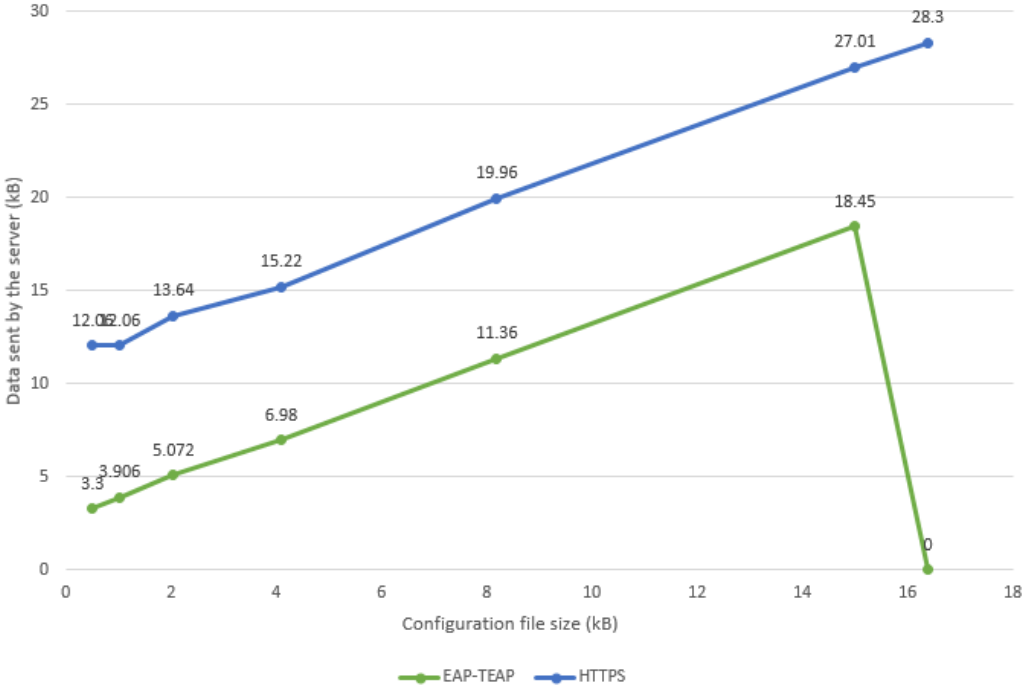

Figure 6.2. Data successfully sent by the server to the client over HTTPS (top line) and in EAP-TEAP (bottom line) as a function of the size of the configuration file.

bigger than 16 kB. The probable reason is that the maximum record size in TLS is 2^{14} bytes, which is 16 kB [16].

In order to remove this restriction, one would have to implement some sort of message fragmentation layer in the vendor-specific extensions of EAP-TEAP. But another way to remove this restriction would be to use the ideas from our solution: instead of the full configuration data, the EAP-TEAP vendor-specific TLV would include only a client token, and the configuration would happen over HTTPS after EAP-TEAP success.

In Figure 6.2, we compare the size of the data sent by the server over HTTPS on the one hand, and in EAP-TEAP on the other hand, as a function of the size of the configuration file. In both cases the data transferred by the server includes (server) parts of the TLS handshake, the provisioning protocol and the configuration protocol.

First, we notice that the server transfers about 8 kB less data in EAP-TEAP than in the HTTPS session. This is because in addition to the configuration data itself, the HTTPS session includes the TLS handshake where the server sends its certificate to the client. We did not try to optimize the HTTPS communication in any way. The provisioning of client certificate using EST opens a new TLS session for each message. As a result, provisioning over HTTPS includes four TLS handshakes, which explains the 8 kB difference.

Second, transferring configuration file of 16.3 kB over EAP-TEAP fails; the fragmentation mechanism in EAP does not allow messages bigger than 16 kB. But the configuration data can

	Time	Source	Destination	Protocol	Length	Info
1	0.000000	Raspberr_85:b0:c9	4e:32:15:c7:46:f7	EAP	28	Request, Identity
2	0.047387	4e:32:15:c7:46:f7	Raspberr_85:b0:c9	EAP	32	Response, Identity
3	0.057106	Raspberr_85:b0:c9	4e:32:15:c7:46:f7	TLSv1.2	34	Ignored Unknown Record
4	0.065852	4e:32:15:c7:46:f7	Raspberr_85:b0:c9	TLSv1.2	212	Client Hello
5	0.090400	Raspberr_85:b0:c9	4e:32:15:c7:46:f7	TLSv1.2	1256	Server Hello, Certificate, Certificate Request, Server Hello Done
6	0.148790	4e:32:15:c7:46:f7	Raspberr_85:b0:c9	EAP	1426	Response, Tunneled EAP protocol
7	0.163056	Raspberr_85:b0:c9	4e:32:15:c7:46:f7	EAP	24	Request, Tunneled EAP protocol
8	1.583578	0.0.0.0	255.255.255.255	DHCP	375	DHCP Discover - Transaction ID 0x919aa94b
9	2.997718	Raspberr_85:b0:c9	4e:32:15:c7:46:f7	EAP	24	Request, Tunneled EAP protocol
10	9.004595	Raspberr_85:b0:c9	4e:32:15:c7:46:f7	EAP	24	Request, Tunneled EAP protocol
11	9.037008	4e:32:15:c7:46:f7	Raspberr_85:b0:c9	TLSv1.2	96	Certificate, Client Key Exchange, Certificate Verify, Change Cipher Spec, Finished
12	9.097095	Raspberr_85:b0:c9	4e:32:15:c7:46:f7	EAP	118	Request, Identity
13	9.106382	4e:32:15:c7:46:f7	Raspberr_85:b0:c9	EAP	86	Response, Identity
14	9.114748	Raspberr_85:b0:c9	4e:32:15:c7:46:f7	EAP	90	Request, MS-Authentication EAP (EAP-MS-AUTH)
15	9.121848	4e:32:15:c7:46:f7	Raspberr_85:b0:c9	EAP	140	Response, MS-Authentication EAP (EAP-MS-AUTH)
16	9.129155	Raspberr_85:b0:c9	4e:32:15:c7:46:f7	EAP	113	Request, MS-Authentication EAP (EAP-MS-AUTH)
17	9.136046	4e:32:15:c7:46:f7	Raspberr_85:b0:c9	EAP	63	Response, MS-Authentication EAP (EAP-MS-AUTH)
18	9.165797	Raspberr_85:b0:c9	4e:32:15:c7:46:f7	EAP	1421	Request, Tunneled EAP protocol
19	9.182220	4e:32:15:c7:46:f7	Raspberr_85:b0:c9	EAP	24	Response, Tunneled EAP protocol
20	9.189597	Raspberr_85:b0:c9	4e:32:15:c7:46:f7	TEAP	574	Intermediate-Result
21	9.205902	4e:32:15:c7:46:f7	Raspberr_85:b0:c9	TEAP	145	Intermediate-Result
22	9.215634	Raspberr_85:b0:c9	4e:32:15:c7:46:f7	EAP	22	Success
23	9.215795	Raspberr_85:b0:c9	4e:32:15:c7:46:f7	EAPOL	113	Key (Message 1 of 4)
24	10.216422	Raspberr_85:b0:c9	4e:32:15:c7:46:f7	EAPOL	113	Key (Message 1 of 4)
25	10.224881	4e:32:15:c7:46:f7	Raspberr_85:b0:c9	EAPOL	137	Key (Message 2 of 4)
26	10.228466	Raspberr_85:b0:c9	4e:32:15:c7:46:f7	EAPOL	137	Key (Message 3 of 4)
27	10.232425	4e:32:15:c7:46:f7	Raspberr_85:b0:c9	EAPOL	113	Key (Message 4 of 4)
28	10.235926	Raspberr_85:b0:c9	4e:32:15:c7:46:f7	EAPOL	137	Key (Group Message 1 of 2)
29	10.241913	4e:32:15:c7:46:f7	Raspberr_85:b0:c9	EAPOL	113	Key (Group Message 2 of 2)

EAP-TEAP

Configuration and provisioning data

Figure 6.3. EAP-TEAP communication captured with Wireshark, modified with the transfer of provisioning and configuration data in the TLS channel.
$Raspberr\_85:b0:c9$ is the server, $4e:32:15:c7:46:f7$ is the client.
Messages 1 to 2: EAP initialization messages
Messages 4 to 11: TLS handshake with server and client certificate
Messages 12 to 17: MSCHAPv2 method in the TLS channel
Messages 18 to 21: agreement on success of authentication, establishment of cryptographic material, transfer of provisioning and configuration data
Message 22: conclusion of EAP communication

easily be larger than 16 kB, for instance if it includes software updates. This restricts the use of EAP-TEAP to relatively small configuration files that are less than 16 kB.

	Time	Source	Destination	Protocol	Length	Info
1	0.000000	Raspberr_85:b0:c9	ba:93:33:68:a5:db	EAP	28	Request, Identity
2	0.115502	ba:93:33:68:a5:db	Raspberr_85:b0:c9	EAP	31	Response, Identity
3	0.121355	Raspberr_85:b0:c9	ba:93:33:68:a5:db	EAP	36	Request, Unknown type (0xc9)
4	0.129990	ba:93:33:68:a5:db	Raspberr_85:b0:c9	EAP	37	Response, Unknown type (0xc9)
5	0.135450	Raspberr_85:b0:c9	ba:93:33:68:a5:db	EAP	33	Request, Unknown type (0xc9)
6	0.143254	ba:93:33:68:a5:db	Raspberr_85:b0:c9	EAP	217	Response, Unknown type (0xc9)
7	0.161580	Raspberr_85:b0:c9	ba:93:33:68:a5:db	EAP	1265	Request, Unknown type (0xc9)
8	0.193427	ba:93:33:68:a5:db	Raspberr_85:b0:c9	EAP	363	Response, Unknown type (0xc9)
9	0.233490	Raspberr_85:b0:c9	ba:93:33:68:a5:db	EAP	84	Request, Unknown type (0xc9)
10	1.653005	0.0.0.0	255.255.255.255	DHCP	375	DHCP Discover - Transaction ID
11	2.999491	Raspberr_85:b0:c9	ba:93:33:68:a5:db	EAP	84	Request, Unknown type (0xc9)
12	3.012186	ba:93:33:68:a5:db	Raspberr_85:b0:c9	EAP	182	Response, Unknown type (0xc9)
13	3.022394	Raspberr_85:b0:c9	ba:93:33:68:a5:db	EAP	117	Request, Unknown type (0xc9)
14	6.003200	Raspberr_85:b0:c9	ba:93:33:68:a5:db	EAP	117	Request, Unknown type (0xc9)
15	6.013809	ba:93:33:68:a5:db	Raspberr_85:b0:c9	EAP	33	Response, Unknown type (0xc9)
16	6.020170	Raspberr_85:b0:c9	ba:93:33:68:a5:db	EAP	37	Request, Unknown type (0xc9)
17	6.025328	ba:93:33:68:a5:db	Raspberr_85:b0:c9	EAP	37	Response, Unknown type (0xc9)
18	6.672856	Raspberr_85:b0:c9	ba:93:33:68:a5:db	EAP	731	Request, Unknown type (0xc9)
19	6.697834	ba:93:33:68:a5:db	Raspberr_85:b0:c9	EAP	37	Response, Unknown type (0xc9)
20	6.702493	Raspberr_85:b0:c9	ba:93:33:68:a5:db	EAP	22	Success
21	6.702642	Raspberr_85:b0:c9	ba:93:33:68:a5:db	EAPOL	113	Key (Message 1 of 4)
22	7.703785	Raspberr_85:b0:c9	ba:93:33:68:a5:db	EAPOL	113	Key (Message 1 of 4)
23	8.704970	Raspberr_85:b0:c9	ba:93:33:68:a5:db	EAPOL	113	Key (Message 1 of 4)
24	9.705472	Raspberr_85:b0:c9	ba:93:33:68:a5:db	EAPOL	113	Key (Message 1 of 4)

Phase 1: EAP-TTLS

Phase 2: EAP-iPROV

Figure 6.4. EAP-oPROV with inner EAP-TTLS and EAP-iPROV communication captured with Wireshark.
Time is in seconds, length is in bytes.
$Raspberr\_85:b0:c9$ is the server, $4e:32:15:c7:46:f7$ is the client.
Messages 3 to 15: phase one of EAP-oPROV, executed with EAP-TTLS.
Messages 16 to 19: phase two of EAP-oPROV, executed with EAP-iPROV.
The client tokens are sent in messages 18.

In Figure 6.3, the configuration and provisioning data are sent after successful client authentication. The structure of the message containing the additional data is described in Figure 6.5.

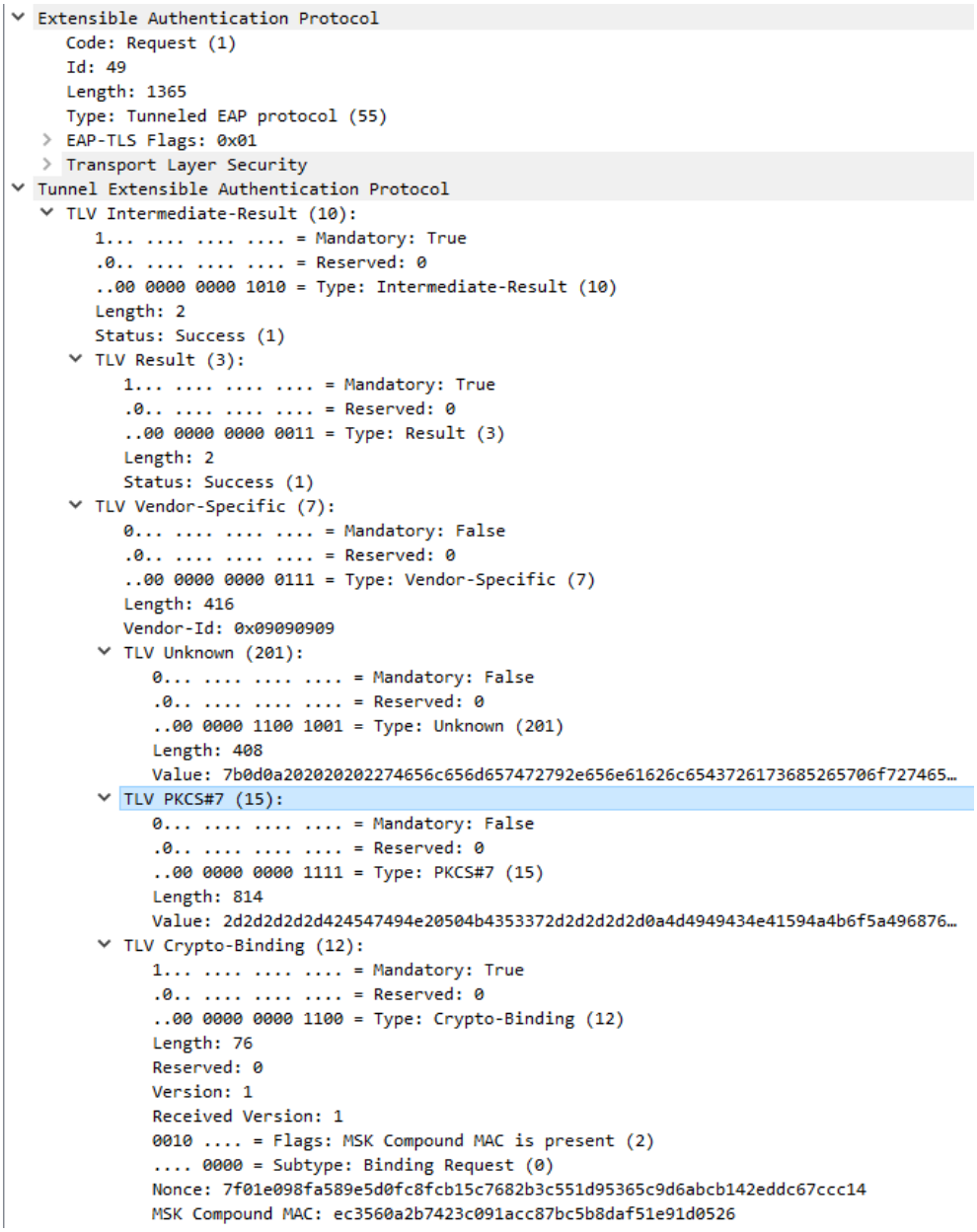

Figure 6.5. Content of the message carrying the provisioning and configuration data
Time is in seconds, length is in bytes.
The “TLV Unknown (201)” contains the configuration data. The “TLV PKCS#7” contains a certificate answering the request of the client previously sent as a PKCS#10 TLV.

The TLV object of type “Vendor-Specific” contains the TLV of type 201, with the configuration data as payload; the TLV of type “PKCS#7” contains the provisioning data. The Vendor-Specific TLV also includes an identifier of the vendor – in this example, the identifier is set to “0x09090909”. The Vendor-Specific TLV is not mandatory: a client that does not understand its content can still authenticate successfully.

6.3 Experiments

We have done a series of experiments to evaluate the latency and communication overhead of different protocol options. First, we varied the size of the configuration data to compare the performance between EAP-TEAP and the use of client tokens. Second, we have fixed the sizes of provisioned certificate and configuration data to about one kilobyte and 400 bytes respectively. The 400 bytes of configuration data could include, for example, network-specific parameters.

In the following, we focus on a small configuration data of about 400 bytes.

(i) As a baseline, we measured the performance of EAP-TEAP with and without provisioning and configuration. A shell script invoked EAP-TEAP repeatedly for 20 minutes, and tcpdump communication records were collected from these runs. EAP-TEAP was configured so that both client and server send certificates to each other during TLS handshake. We run EAP-TEAP both with and without provisioning and configuration. The results of this experiment are summarized in rows (0) and (1) of Table 6.2.

(ii) Next, a similar script repeatedly invoked EAP-oPROV with inner methods EAP-TTLS (for mutual authentication) and EAP-iPROV (for sending provisioning and configuration tokens), followed by provisioning and configuration over HTTPS. EAP-TTLS was configured so that both client send certificates to each other during TLS handshake. The results of this experiment are summarized in rows (2) and (7) of Table 6.2.

(iii) The measurements were repeated for EAP-TTLS with EAP-iPROV followed by provisioning and configuration over HTTPS. The results of this experiment are summarized in rows (3) and (8) of Table 6.2.

All in all, about 100 runs were done for each protocol option during the experiment. We wrote a Python script to filter and compute statistics from these records. Table 6.2 summarizes the results.

The first column of the table identifies the protocol option whose performance we have measured. For brevity, we will use the number of protocol option instead of its full name when discussing the table entries below. For example, we write “protocol (1)”, instead of “EAP-TEAP+provisioning and configuration”. The other columns of the table contain the number of messages and amount of data sent by client and server. If there was variation in those numbers between protocol runs, we provide the median and the extremes in square brackets across all protocol runs. For example, “10 [10, 11]” in row (3), second column, means that the median, minimum and maximum number of messages sent by the server was 10, 10 and 11 respectively.

The table is divided horizontally into three blocks. The first block (protocols (0), (1), (2) and (3)) includes EAP messages only. The second block (protocols (4), (5) and (6)) includes the HTTP communication for provisioning and configuration after successful completion of the EAP method in protocols (2) and (3). Please note that these HTTP communications have not been optimized: the client creates a new TLS session with the server for each EST protocol request. The third block (protocols (7) and (8)) includes both the EAP communication and the subsequent HTTP communication, to obtain total cost for authentication, provisioning and configuration.

Looking at Table 6.2, we see that:

- Comparing protocol (0) and (1) we see that adding provisioning and configuration to EAP-TEAP authentication adds about 2 kB, and less than 0.1 s.

- Protocol (1) takes roughly 0.5 s, while protocols (2) and (3) take about one second. The reason for the additional time in the latter is the initialization of the two EAP methods: EAP-TTLS and EAP-iPROV.

- The number of messages in protocols (1), (2), and (3) is about 10. Note that (3) has one message and 32 bytes more than than (2), and it takes 0.04 second more time. This is because there is one extra EAP Hello message in EAP-oPROV.

- Less data is exchanged in protocol (2) or (3) than in (1), because in (2) and (3) the actual provisioning and configuration happen outside of the EAP exchange.

- Protocol (4) has more messages and takes longer than (5), because the EST protocol used in (4) includes several round trips, while (5) is completed within a single round trip.

- The time for running (4) and (5) is about 0.3 second; but (6) takes about 0.9 second. The reason is that in (6), the operations (4) and (5) are run inside a Python script in the IoT device, which adds the overheads of context switching and compilation.

- The transfer of configuration and provisioning data over HTTP adds about 14 kB. This overhead could be reduced. For example, since each TLS handshake involves exchange of about two kilobytes, reducing the number of TLS handshakes from three to one in EST provisioning, would reduce the amount data by about four kilobytes. We leave these optimizations to future work.

All in all, in our experiments provisioning and configuration with EAP-TEAP is faster (by about 0.5 s) than EAP-oPROV/EAP-iPROV. The amount of data exchanged inside the EAP-TEAP is larger (by about 600 B) than that inside EAP-oPROV/EAP-iPROV. The reason is that in case of EAP-oPROV/EAP-iPROV, the actual provisioning and configuration happen over HTTPS, after EAP-oPROV success.

	Messages from server	Bytes from server	Messages from client	bytes from client	duration (seconds)
0. EAP-TEAP without provisioning or configuration	10	1862	9	2196	0.42 [0.302, 0.659]
1. EAP-TEAP+provisioning and configuration	10	3092	9	2989	0.4635 [0.367, 0.621]
2. EAP-TTLS/EAP-iPROV	10 [10, 11]	2400 [2392, 3177]	9 [9, 10]	2094 [2094, 2167]	0.943 [0.856, 1.094]
3. EAP-oPROV/EAP-TTLS+ EAP-iPROV	11 [11, 12]	2432 [2416, 3179]	10 [10, 11]	2150 [2150, 2193]	0.981 [0.879, 1.148]
4. HTTP provisioning	15	9594	9	3257 [3256, 3257]	0.23 [0.22, 0.34]
5. HTTP configuration	7	3875 [3874, 3876]	5	1759	0.03 [0.029, 0.031]
6. HTTP communications (both provisioning and configuration)	22 [15, 23]	13468 [10260, 14918]	14 [10, 14]	5016 [3343, 5016]	0.88 [0.83, 1.18]
7. EAP-TTLS/EAP-iPROV +HTTP communications	32 [25, 34]	15868 [12452, 18095]	23 [19, 24]	7110 [5337, 7183]	1.823 [1.686, 2.274]
8. EAP-oPROV/EAP-TTLS+EAP-iPROV +HTTP communications	33 [26, 35]	15900 [14676, 18097]	24 [20, 25]	7166 [5493, 9376]	1.69 [1.709, 2.328]

Table 6.2. Summary of experimental measurements of provisioning and configuration with EAP-TEAP, EAP-oPROV and EAP-iPROV with EAP-TTLS as inner method.
Each cell contains the median value, followed by the segment [minimum, maximum] values when different values have been recorded.

Recall that EAP-oPROV runs first an inner EAP method (in phase one) and then EAP-iPROV inside its session. It (only) requires mutual authentication and session key from the inner EAP method (in phase one). Moreover, EAP-oPROV is extendable, because the authentication server has full control over which inner EAP methods are run inside EAP-oPROV. For example, the authentication server can choose to run more than one inner EAP method inside EAP-oPROV. Also, the authentication server may choose not to run EAP-iPROV at all, i.e. EAP-oPROV can be used to combine different EAP methods. Thus, EAP-oPROV is a generic method for extending EAP functionality. It could be applied in scenarios other than provisioning and configuration.

EAP-iPROV can be executed not only inside EAP-oPROV, but inside EAP-TTLS for instance. EAP-iPROV can be potentially used to exchange any information that is short enough to fit in a single packet. However, this requires specification of the intended usage of that information.

In conclusion, using the generic method EAP-oPROV for provisioning and configuration is less efficient than adding provisioning and configuration functionality into one specific method (EAP-TEAP), but it is more widely applicable because it can be combined with many EAP methods.

6.4 Future work

Future work could include the following.

- Defining support for fragmentation in EAP-oPROV. Indeed, EAP-oPROV messages add a

header are slightly bigger than the messages of the inner methods because of the addition of a header. The inner method messages are smaller than the MTU, and the addition of the header can lead to an EAP-oPROV message bigger than the MTU. This rare situation must be avoided: providing fragmentation in EAP-oPROV is a solution, fragmentation in EAP being easy to implement thanks to its structure of request and response.

- Defining the client tokens that can be used in EAP-iPROV (it may be advantageous to define client tokens for different types of configuration). It is necessary to define all types of client tokens that could be necessary to allow large adoption of EAP-iPROV.

- Writing specification for both methods EAP-oPROV and EAP-iPROV, and submitting it to IETF.

- Adding EAP-oPROV and EAP-iPROV to EAP implementations.

- Implementing a solution for bootstrapping an IoT device that does not yet have long-term credentials. The solution would have both (i) initial identification of the device using an ad-hoc or a hybrid method, [11]. and (ii) provisioning and configuration with EAP-oPROV and EAP-iPROV. For example, the initial identification could be done using EAP-NOOB.

7. Conclusion

EAP is widely used for peer authentication in wireless network. On the other hand its use for client management is limited to EAP-TEAP [2] and EAP-CREDS, which is an internet draft. In both cases the EAP method includes (only) the provisioning of a credential (e.g., certificate) to the client and this operation is done entirely inside an EAP session.

In this thesis we consider a different approach, where we add support in EAP method for transferring client tokens from the authentication server. A client token includes the URL of the management server, the client identity signed by the authentication server and a timestamp. The actual client management operations, like provisioning and configuration, happen over HTTPS after successful EAP authentication; the client uses the tokens to access the management servers.

This approach can be implemented with rather small changes in the EAP method and without any changes in the WiFi access point (EAP authenticator). On the other hand, it entails some sort of separation of devices that have the client tokens, but have not yet been provisioned and configured. This separation can be done on application, IP or data link layers.

We have designed the following four options for transferring client tokens inside an EAP session. These options were then implemented by extending the open-source hostapd and wpa_supplicant software components in Raspberry Pi.

1. EAP notification message;
2. Additional attribute in existing EAP method's message;
3. EAP-iPROV run as an inner method in TLS tunnel;
4. EAP-oPROV: an outer EAP method that first authenticates the peer using some inner EAP method, and then runs EAP-iPROV.

Based on our analysis and experiments, option (4) gives the most flexibility for adding client tokens in EAP. Moreover, EAP-oPROV is a generic method for extending EAP functionality. It

could be applied in scenarios other than provisioning and configuration.

Future work could include standardization of EAP-oPROV and finding additional scenarios were EAP-oPROV can be used, like combining authentication methods for achieving stronger security.

Appendices

A. Abbreviations

AAA	Authentication, Authorization and Accounting
AAD	associated additional data
ACK	Acknowledgement
AEAD	Authenticated Encryption with Associated Data
AES	Advanced Encryption Standard
AES-GCM	Advanced Encryption Standard - Galois Counter Mode
API	Application Programming Interface
AP	Access Point
AVP	Attribute Value Pair
DHCP	Dynamic Host Configuration Protocol
EAP	Extensible Authentication Protocol
EAP-AKA	EAP with Authentication and Key Agreement
EAP-CREDS	EAP Credentials Provisioning and Management
EAP-FAST	EAP with Flexible Authentication via Secure Tunneling
EAP-LEAP	EAP with Lightweight EAP
EAP-PEAP	Protected EAP
EAP-PSK	EAP with Pre-Shared Key
EAP-SIM	EAP with Subscriber Identity Module
EAP-TEAP	EAP with Tunneled EAP
EAP-TLS	EAP with Transport Layer Security
EAP-TTLS	EAP with Tunneled Transfer Layer Security
EAPOL	EAP Over LAN
ECDSA	Elliptic Curve Digital Signature Algorithm
EST	Enrollment over Secure Transport
ES256	SHA256 with ECDSA
GCS	Ground Control Station
GTC	Generic Token Card

HTTP	HyperText Transfer Protocol
HTTPS	Secure HTTP
IoT	Internet of Things
IP	Internet Protocol
IV	Initialization Vector
LAN	Local Access Network
LTE 4G	Long Term Evolution fourth generation
MD5	Message Digest 5
MIC	Message Integrity Check
MSCHAP	Microsoft extension to Challenge Handshake Authentication Protocol
MSCHAPv2	MSCHAP version 2
MSK	Master Session Key
NAI	Network Access Identifier
OOB	Out-Of-band
OTP	One Time Password
OS	Operating System
PAC	Protected Access Credential
PIN	Personal Identification Number
PPP	point-to-point
PKI	Public Key Infrastructure
QR	Quick Response
RADIUS	Remote Authentication Dial In User Service
RAM	Random Access Memory
REST	Representational State Transfer
RFC	Request For Comments
RJ45	Registered Jack 45
SHA256	Secure Hash Algorithms 256 bits
SSID	Service Set Identifier
SSH	Secure Shell
TLV	Type Length Value (object format)
UAV	Unmanned Aerial Vehicle
UTF8	Unicode Transformation Format 8
URL	Uniform Resource Locator
VLAN	Virtual Local Access Network
VNF	Virtual Network Function
WiFi	Wireless Fidelity
WiFi AP	WiFi Access Point

WPA	WiFi Protected Access
WPS	WiFi Protected Setup

B. Additional information about EAP methods

The EAP methods commonly used in WiFi networks are EAP-TTLS, EAP-FAST, EAP-PEAP, EAP-LEAP and EAP-TLS [34]. For instance, Eduroam [8] provides support for EAP-TLS, EAP-TTLS and EAP-PEAP. The WiFi Protected Access 2 (WPA2) standard recommends these EAP methods with an addition of EAP-FAST. Table 2.1 shows that, with an exception of EAP-FAST, these EAP methods are supported in Android, Linux and Windows operating systems; EAP-FAST is currently supported only in Linux.

EAP method	Android	Linux	Windows
EAP-TTLS	+	+	+
EAP-TLS	+	+	+
EAP-PEAP	+	+	+
EAP-AKA	+	+	+
EAP-SIM	+	+	+
EAP-FAST	-	+	-

Table 2.1. Support for EAP methods recommended in WPA2 [35][36][37].
support: “+”, no support: “-”

Table 2.2 describes the characteristics provided by the EAP methods mentioned in this thesis.

	fast-reconnection	fragmentation support	key material sharing	authentication type
EAP-PWD	no	no	yes	mutual
EAP-POTP	no	no	yes	client or mutual
EAP-SIM	yes	no	yes	mutual
EAP-AKA’	yes	no	yes	mutual
EAP-TLS	yes	yes	yes	mutual
EAP-TTLS	yes	yes	yes	mutual
EAP-TEAP	yes	yes	yes	mutual
EAP-PEAP	yes	yes	yes	mutual
EAP-FAST	yes	yes	yes	mutual
EAP-CREDS	no	no	no	none

Table 2.2. Properties of EAP methods

In the rest of this Appendix we list EAP methods that can be used with EAP-oPROV. The

methods in this list where selected using the following criteria:

- The method includes mutual authentication resulting in session key.
- There is a public specification of the method.
- The source code is publicly available.

B.0.1 EAP-PWD

EAP-PWD (Password) [23] was initially developed for secure connection of a user on a computer with a password stored on a remote server. EAP-PWD allows creation of high entropy key material from the low entropy password. EAP-PWD requires three message exchanges:

- Identity message: it provides the identity of the user
- Challenge message: it provides data for verifying the password
- Confirmation message: it provides the response to the previous challenge

EAP-PWD does not support message fragmentation. A successful EAP-PWD authentication results in sharing of key material between the peer and the authentication server.

B.0.2 EAP-POTP

EAP-POTP (Protected One-Time Password) [24] was initially developed for OTP (One-Time Password) tokens electronically connected to the authenticating computer. It provides one-way or two-way authentication. EAP-POTP provides a framework for the execution of a specific type of credential, a shared secret key, and supports multiple algorithms to perform the challenges verifying the shared secret. EAP-POTP uses TLV as attribute format and protects its content with encryption authenticity verification.

A successful EAP-POTP authentication results in sharing of key material between the peer and the authentication server.

B.0.3 EAP-SIM

EAP-SIM [14] (Subscriber Identity Modules) was initially developed as a second generation mobile network standard. It allows server or mutual authentication and fast re-authentication.

EAP-SIM does not support fragmentation. A successful EAP-SIM authentication results in sharing of key material between the peer and the authentication server.

B.0.4 EAP-AKA'

EAP-AKA' (Authentication and Key Agreement) [22] is an EAP method for LTE(4G). It allows mutual authentication. The attributes in the EAP payloads are in TLV format. The steps of the protocol are:

- phase one: receiving the identity of the client. The authenticator receives the identity and verifies the validity by contacting the correct authentication server. If successful, the authentication server returns a vector allowing verification of the identity.
- phase two: the authenticator sends a challenge to the peer based on the received vector. The peer can determine the authenticity of the authenticator by verifying the correctness of the challenge, which requires the knowledge of the vector sent by the authentication server. The peer replies with a challenge response.

EAP-AKA' allows fast reauthentication and accepts extensions with skippable attributes in the payloads. It does not support fragmentation: all messages should be smaller than 1020 bytes – according to the specification. A successful EAP-AKA' authentication results in sharing of key material between the peer and the authentication server.

Please note that EAP-AKA' (and EAP-SIM) require the client to have a SIM card, i.e. the client has to have cellular network subscription.

B.0.5 EAP-PEAP

EAP-PEAP (Protected Authentication Protocol) [15] is a mutual authentication method initially developed for wireless networks. The steps of the protocol are:

- phase one: the peer and authentication server involve in a TLS handshake. This step provides server authentication and a secure communication channel.
- phase two: this phase executes another EAP method inside EAP-PEAP messages. The inner EAP method provides client authentication.

EAP-PEAP is close from EAP-TTLS. A successful EAP-PEAP authentication results in sharing of key material between the peer and the authentication server.

B.0.6 EAP-FAST

EAP-FAST (Flexible Authentication via Secure Tunneling) [4] performs server authentication and optional client authentication. It is meant to be lightweight for wireless network improvement. The steps of the protocol are:

- phase one: a TLS handshake provides server authentication and a secure communication

channel

- phase two: inner EAP methods can be run sequentially as TLV attributes. The purpose is client authentication and key material setting.

EAP-FAST supports fragmentation and provisioning of a Protected Access Credential (PAC). A successful EAP-FAST authentication results in sharing of key material between the peer and the authentication server.

Bibliography

[1] John Vollbrecht, James D. Carlson, Larry Blunk, Dr. Bernard D. Aboba, and Henrik Levkowetz, "Extensible Authentication Protocol (EAP)," RFC 3748, Jun. 2004. [Online]. Available: https://rfc-editor.org/rfc/rfc3748.txt

[2] Hao Zhou, Nancy Cam-Winget, Joseph A. Salowey, and Steve Hanna, "Tunnel Extensible Authentication Protocol (TEAP) Version 1," RFC 7170, May 2014. [Online]. Available: https://rfc-editor.org/rfc/rfc7170.txt

[3] Paul Funk and Simon Blake-Wilson, "Extensible Authentication Protocol Tunneled Transport Layer Security Authenticated Protocol Version 0 (EAP-TTLSv0)," RFC 5281, Aug. 2008. [Online]. Available: https://rfc-editor.org/rfc/rfc5281.txt

[4] Joseph A. Salowey, Hao Zhou, Nancy Cam-Winget, and David McGrew, "The Flexible Authentication via Secure Tunneling Extensible Authentication Protocol Method (EAP-FAST)," RFC 4851, May 2007. [Online]. Available: https://rfc-editor.org/rfc/rfc4851.txt

[5] Jodi Haasz, "IEEE Standard for information technology-Telecommunications and information exchange between systems-Local and metropolitan area networks-Specific requirements-Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Amendment 6: Medium Access Control (MAC) Security Enhancements," *IEEE Std 802.11i-2004*, pp. 1–190, 2004.

[6] Massimiliano Pala, "Credentials Provisioning and Management via EAP (EAP-CREDS)," Internet Engineering Task Force, Internet-Draft draft-pala-eap-creds-07, Jun. 2020, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-pala-eap-creds-07

[7] Tolgahan Akgun, "Secure Lifecycle Management for Internet of Things Devices, MSc thesis, Aalto University," 2020. [Online]. Available: https://aaltodoc.aalto.fi/handle/123456789/46142

[8] Anyroam, EAP Method Feature Comparison. [Online]. Available: https://www.anyroam.net/node/80

[9] Tuomas Aura and Mohit Sethi, "Nimble out-of-band authentication for EAP (EAP-NOOB)," Internet Engineering Task Force, Internet-Draft draft-aura-eap-noob-08, Mar. 2020, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-aura-eap-noob-08

[10] Pietro Boccadoro, Domenico Striccoli, and Luigi Alfredo Grieco, "An Extensive Survey on the Internet of Drones," 2020.

[11] Mohit Sethi, Behcet Sarikaya, and Dan Garcia-Carrillo, "Secure IoT Bootstrapping: A Survey," Internet Engineering Task Force, Internet-Draft draft-sarikaya-t2trg-sbootstrapping-11, Feb. 2021, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-sarikaya-t2trg-sbootstrapping-11

[12] John Vollbrecht and Larry Blunk, “PPP Extensible Authentication Protocol (EAP),” RFC 2284, Mar. 1998. [Online]. Available: https://rfc-editor.org/rfc/rfc2284.txt

[13] Daniel Simon, Ryan Hurst, and Dr. Bernard D. Aboba, “The EAP-TLS Authentication Protocol,” RFC 5216, Mar. 2008. [Online]. Available: https://rfc-editor.org/rfc/rfc5216.txt

[14] Joseph A. Salowey and Henry Haverinen, “Extensible Authentication Protocol Method for Global System for Mobile Communications (GSM) Subscriber Identity Modules (EAP-SIM),” RFC 4186, Jan. 2006. [Online]. Available: https://rfc-editor.org/rfc/rfc4186.txt

[15] Ashwin Palekar, Simon Josefsson, Daniel Simon, and Glen Zorn, “Protected EAP Protocol (PEAP) Version 2,” Internet Engineering Task Force, Internet-Draft draft-josefsson-pppext-eap-tls-eap-10, Oct. 2004, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-josefsson-pppext-eap-tls-eap-10

[16] Eric Rescorla and Tim Dierks, “The Transport Layer Security (TLS) Protocol Version 1.2,” RFC 5246, Aug. 2008. [Online]. Available: https://rfc-editor.org/rfc/rfc5246.txt

[17] Massimiliano Pala. EAP-CREDS: Enabling Policy-Oriented Credential Management in Access Networks. [Online]. Available: https://www.cablelabs.com/eap-creds-enabling-policy-oriented-credential-management-in-access-networks#:~:text=For%20the%20business%20and%20industrial,regularly%20updated%20and%20(c)%20credentials

[18] O. Arias, F. Rahman, M. Tehranipoor, and Y. Jin, “Device attestation: Past, present, and future,” in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2018, pp. 473–478.

[19] Michael Jones, John Bradley, and Nat Sakimura, “JSON Web Token (JWT),” RFC 7519, May 2015. [Online]. Available: https://rfc-editor.org/rfc/rfc7519.txt

[20] Farid Adrangi, Farooq Bari, Pasi Eronen, and Victor Lortz, “Identity Selection Hints for the Extensible Authentication Protocol (EAP),” RFC 4284, Jan. 2006. [Online]. Available: https://rfc-editor.org/rfc/rfc4284.txt

[21] Nick L. Petroni, John Vollbrecht, Yoshihiro Ohba, and Pasi Eronen, “State Machines for Extensible Authentication Protocol (EAP) Peer and Authenticator,” RFC 4137, Aug. 2005. [Online]. Available: https://rfc-editor.org/rfc/rfc4137.txt

[22] Jari Arkko, Vesa Lehtovirta, and Pasi Eronen, “Improved Extensible Authentication Protocol Method for 3rd Generation Authentication and Key Agreement (EAP-AKA’),” RFC 5448, May 2009. [Online]. Available: https://rfc-editor.org/rfc/rfc5448.txt

[23] Glen Zorn and Dan Harkins, “Extensible Authentication Protocol (EAP) Authentication Using Only a Password,” RFC 5931, Aug. 2010. [Online]. Available: https://rfc-editor.org/rfc/rfc5931.txt

[24] Magnus Nystrom, “The EAP Protected One-Time Password Protocol (EAP-POTP),” RFC 4793, Feb. 2007. [Online]. Available: https://rfc-editor.org/rfc/rfc4793.txt

[25] Romain Berrendonner and Herve Chabanne, “PPP EAP MAKE Mutual Authentication Protocol,” Internet Engineering Task Force, Internet-Draft draft-berrendo-chabanne-pppext-eapmake-01, Nov. 2001, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-berrendo-chabanne-pppext-eapmake-01

[26] Jouni Malinen. (2013) hostapd and wpa supplicant. [Online]. Available: https://w1.fi/

[27] Max Pritikin, Peter E. Yee, and Dan Harkins, “Enrollment over Secure Transport,” RFC 7030, Oct. 2013. [Online]. Available: https://rfc-editor.org/rfc/rfc7030.txt

[28] abrox. Simplest Git repository for EST provisioning. [Online]. Available: https://github.com/abrox/simplest

[29] Paul Funk and Simon Blake-Wilson, “EAP Tunneled TLS Authentication Protocol Version 1 (EAP-TTLSv1) ,” Internet Engineering Task Force, Internet-Draft draft-funk-eap-ttls-v1-01, Mar. 2006, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-funk-eap-ttls-v1-01

[30] Ashwin Palekar, Simon Josefsson, Daniel Simon, and Glen Zorn, “Protected EAP Protocol (PEAP) Version 2,” Internet Engineering Task Force, Internet-Draft draft-josefsson-pppext-eap-tls-eap-10, Oct. 2004, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-josefsson-pppext-eap-tls-eap-10

[31] Jose Puthenkulam, “The Compound Authentication Binding Problem,” Internet Engineering Task Force, Internet-Draft draft-puthenkulam-eap-binding-04, Oct. 2003, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-puthenkulam-eap-binding-04

[32] Morris Dworkin, Elaine Barker, James Nechvatal, James Foti, Lawrence Bassham, E. Roback, and James Dray, “Advanced Encryption Standard (AES),” 2001-11-26 2001.

[33] Scott A. Vanstone Alfred J. Menezes, Paul C. van Oorschot, *Handbook of applied cryptography*, ser. International series of monographs on physics. CRC Press, October 1996, p. 53.

[34] “WPA3 Specification Version 3.0,” Tech. Rep., 2020. [Online]. Available: https://www.wi-fi.org/download.php?file=/sites/default/files/private/WPA3_Specification_v3.0.pdf

[35] Windows documentation. EAP configuration. [Online]. Available: https://docs.microsoft.com/en-us/windows/client-management/mdm/eap-configuration

[36] Linux documentation. Linux support. [Online]. Available: https://www.nowiressecurity.com/configure-8021x-authentication-linux

[37] Android documentation. EAP support. [Online]. Available: https://developer.android.com/reference/android/net/wifi/WifiEnterpriseConfig.Eap