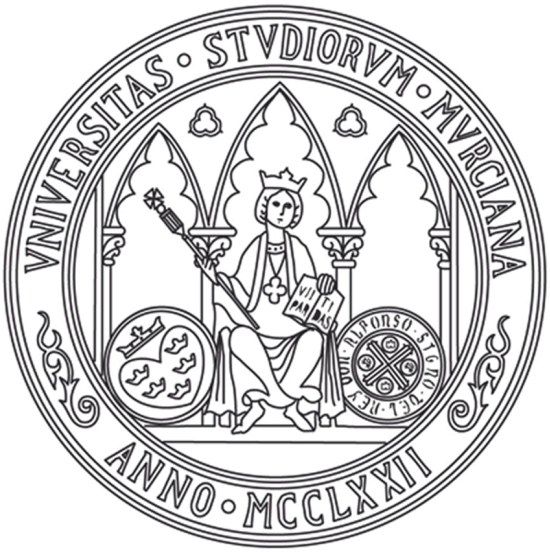


UNIVERSIDAD DE MURCIA

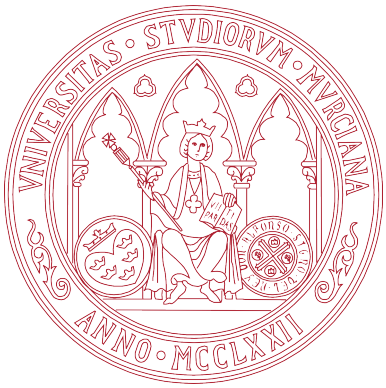
ESCUELA INTERNACIONAL DE DOCTORADO

A CoAP-Based Bootstrapping Service for Large-Scale Internet-of-Things Networks

Un Servicio de Bootstrapping Basado en CoAP para Redes a Gran Escala de Internet de las Cosas

D. Dan García Carrillo

2018



Universidad de Murcia

Facultad de Informática

Un Servicio de Bootstrapping Basado en CoAP para Redes a Gran Escala de Internet de las Cosas

TESIS DOCTORAL

Presentada por:
Dan García Carrillo

Supervisada por:
Prof. Dr. Rafael Marín López
CTO Odin Solutions Prof. Dr. Antonio F. Skarmeta Gómez

Murcia, Julio de 2018

Resumen

El Internet de las cosas (IoT por sus siglas en inglés) se refiere a la interconexión de los dispositivos informáticos integrados en objetos cotidianos a través de Internet, lo que les permite transmitir datos, tal como lo define el Diccionario de Oxford en su versión en linea. Estos dispositivos integrados generalmente se llaman objetos inteligentes (smart objects) debido a su capacidad para interactuar con su entorno. Pueden ser sensores o actuadores si se reúnen información o realizar alguna acción física, respectivamente. El IoT provee, a través de smart objects, servicios avanzados para gestionar de manera eficiente las infraestructuras, como hogares inteligentes, edificios y ciudades inteligentes, etc. Así como o los procesos productivos de una empresa, como el mantenimiento y ubicación de su inventario.

El IoT tiene el potencial de tener un impacto positivo en la economía y la sociedad en general. Aplicaciones tales como personas monitorizando su salud y bienestar, mejorando el proceso productivo de una empresa, controlando el consumo de energía de una población a través de una "Smart Grid", etc. Todo esto es posible gracias a la conexión de objetos inteligentes a Internet.

El interés en esta área está creciendo, no solo en la comunidad investigadora, sino también en empresas y organizaciones de estandarización debido a su potencial. De hecho, hay un trabajo hecho, así como en curso, para proporcionar conectividad IP a la gran cantidad de dispositivos que se espera sean parte del IoT.

Entre las diferentes áreas de investigación y desarrollo relacionadas con el Internet de las cosas, hay uno de particular interés, que es la Seguridad. La razón es que los objetos inteligentes tratan con información sensible. En particular, existe la preocupación de proporcionar los medios necesarios para proteger las comunicaciones, no solo desde el punto de vista de la información que puede obtenerse, sino que además, por las posibles repercusiones realizadas por dispositivos como actuadores, que tienen un impacto y repercusión rea.

Proteger las comunicaciones IoT es un desafío, ya que involucran objetos inteligentes con un conjunto de características y limitaciones tales como las tecnologías de radio utilizadas, así como diferentes restricciones en la Unidad Central de Procesamiento (CPU), memoria, consumo de energía y ancho de banda. Esto hace que sea más difícil proteger las comunicaciones, porque los protocolos utilizados en los sistemas convencionales pueden no

ser aplicables, teniendo que adaptarse o rediseñarse, en algunos casos, dichos protocolos para proteger el IoT.

Hay acuerdo en que, para asegurar las comunicaciones en el IoT, necesitamos ciertos procesos de seguridad, como la autenticación (para determinar la identidad del dispositivo), la autorización (para determinar qué puede hacer el dispositivo) y la clave. gestión (que proporciona material clave para asegurar las comunicaciones). Estos procesos de seguridad están presentes en varios casos de uso, como para acceder a un servicio, establecer una comunicación segura entre dos partes, y también son parte de lo que se llama bootstrapping.

Bootstrapping es el proceso que establece las bases para que un smart object se una de forma segura a una red y pueda proteger sus comunicaciones, convirtiéndose en una parte confiable dentro de un dominio de seguridad. En general, una solución de Bootstrapping para IoT supone que va a haber un smart object que intenta unirse a una red formada por varios objetos inteligentes, que conforma la parte restringida de la red. La red es parte del dominio de seguridad de un Controlador, que es un dispositivo que no está tan limitado como los smart objects. Para realizar el bootstrapping ambas entidades se comunicarán a través de un protocolo de bootstrapping, lo que implica el intercambio de mensajes del protocolo de bootstrapping, con el objetivo de establecer una asociación de seguridad inicial (SA por sus siglas en inglés) entre el smart object y el Controlador, lo que permite que el smart object se convierta una parte confiable en el dominio de seguridad. Debido a que el protocolo de bootstrapping se ejecuta en un enlace restringido, necesita ser lo más ligero posible.

Actualmente, hay protocolos de bootstrapping que se utilizan en el contexto de IoT, pero generalmente asumen que los dispositivos van a ser administrados bajo el dominio de una organización con un solo Controlador, sin tener en cuenta que el smart object podría cambiar de Controlador, moviéndose a un dominio de seguridad diferente, o que smart objects de otras organizaciones podría ser instalados también. Identificamos esas implementaciones como de pequeñas o media escala, donde no hay necesidad de usar administración avanzada para hacer el bootstrapping a un gran número de dispositivos, considerando diferentes organizaciones. Algunos otros protocolos de bootstrapping están diseñados para una tecnología específica de capa de enlace y no tienen en cuenta la diversidad de tecnologías en implementaciones a gran escala, como en el caso de un campus universitario o una Smart City, con diferentes edificios, áreas verdes, estacionamientos, etc. todas las instalaciones, tal vez, pertenecientes a un dominio administrativo diferente. En otros casos, los protocolos de bootstrapping no han sido diseñados teniendo en cuenta las diferentes limitaciones de IoT. Estas son las razones por las que identificamos unas deficiencias importantes en bootstrapping en IoT, ya que las soluciones actuales de bootstrapping, generalmente no tienen en cuenta despliegues a gran escala, y si lo hacen, no son adecuadas para dispositivos y redes restringidos.

Hay ciertos requisitos que identificamos en un servicio de bootstrapping para despliegues a gran escala. Debería funcionar independientemente de la capa de enlace y tener en cuenta el objetivo de hacer ligero protocolo de bootstrapping. Esto podría lograrse intentando reutilizar el código tanto como sea posible, además de reducir el número de bytes enviados a través de la red.

Debido a la posibilidad de tener diferentes tecnologías de capa de enlace, la solución de bootstrapping necesita proporcionar gestión flexible de claves para ejecutar el protocolo de asociación de seguridad que mejor se adapte a la implementación específica. También necesita tener en cuenta la posibilidad de tener dispositivos de diferentes organizaciones, por lo que necesitaríamos federación de identidades. Se requiere autenticación flexible para suportar mecanismos de autenticación diferentes. Esto puede ser necesario dependiendo de las implementaciones y despliegues que se realicen. Ejemplo de esto sería qué smart objects se usan, qué capacidad y restricciones tiene en el enlace, etc. y qué políticas tiene la empresa que gestiona los smart objects, puesto que pueden requerir el uso de un método de autenticación específico. Para proporcionar información para el arranque y la asociación de seguridad necesitamos autorización flexible para enviar los parámetros de autorización necesarios (por ejemplo, credenciales, duración del bootstrapping, etc.). Además, la contabilidad es necesaria para controlar el uso de los recursos de la red. Finalmente, necesitamos soportar diferentes topologías, para las cuales las redes multisalto deben ser compatibles con la solución de bootstrapping. En esta disertación, profundizamos en el área de bootstrapping en el Internet de las Cosas para proporcionar un servicio de bootstrapping para despliegues a gran escala que tenga en cuenta los anteriores requisitos. Para lograr esto, proporcionamos tres contribuciones principales en esta tesis.

CoAP-EAP es un servicio de arranque construido sobre 3 pilares: Infraestructuras de "Authentication Authorization and Accounting" (AAA), el protocolo "Extensible Authentication Protocol" (EAP) y el protocolo "Constrained Application Protocol" (CoAP). Presentamos la arquitectura y el flujo general de operación. CoAP-EAP ofrece un servicio de bootstrapping que es independiente de la capa de enlace. Utiliza CoAP como transporte para EAP (implementa un "EAP lower layer" en terminología de EAP), reduciendo la sobrecarga en comparación con otros EAP lower layer utilizados actualmente en el IoT (como el protocolo PANA), y al mismo tiempo reutiliza código, utilizando CoAP que estará presente en una gran cantidad de smart objects que proporcionan servicios en el IoT. Esta contribución se prueba en una red inalámbrica de área personal de baja velocidad (LR-WPAN), como representante de Wireless Personal Area Network (WPAN).

LO-CoAP-EAP se presenta como una consecuencia de la relevancia obtenida, en los últimos años, por un nuevo conjunto de tecnologías de radio, llamado Low Power Wide Area

Networks (LPWAN). Este conjunto de tecnologías ofrece comunicaciones de largo alcance, enviando pocos bytes con un consumo de energía reducido. LPWAN tiene altas restricciones en el enlace, lo que requiere un rediseño del servicio de bootstrapping, teniendo en cuenta estas limitaciones, para proporcionar un servicio de bootstrapping que es independiente de la tecnología de capa de enlace. LO-CoAP-EAP proporciona una alternativa ligera que agrega flexibilidad al proceso de bootstrapping en LPWAN, logrando una compensación entre rendimiento y flexibilidad, además de proporcionar soporte de federación de identidades en LPWAN. Presentamos la arquitectura y el flujo general de operación, y probamos LO-CoAP-EAP utilizando la tecnología de radio de largo alcance (LoRa) y una red LoRaFabian en un despliegue real en Rennes, Francia. Esta contribución también se prueba en un LR-WPAN para verificar que la mejora también se puede aprovechar en este tipo de redes.

En la última contribución principal, consideramos el caso de las redes multisalto donde, para realizar el bootstrapping, un smart object puede no ser capaz de contactar a la entidad a cargo de la autenticación (Controlador) directamente. En este caso, el smart object tiene que depender de otra entidad para que le ayude en el proceso de bootstrapping. Esto puede deberse al hecho de que el smart object no tiene un IP enrutable hasta que se complete el bootstrapping, o no se puede alcanzar por radio al Controlador. Para cubrir estas posibilidades, definimos una entidad intermediaria, que se agrega a la arquitectura del servicio de bootstrapping. Esta entidad se instancia en tres variantes. Un proxy, que se basa en la entidad proxy de CoAP definida en el estándar CoAP. Esta entidad permite el análisis y la modificación de mensajes, así como mantener el estado relacionado al intercambio de mensajes. Con el fin de tener una entidad que no tiene que analizar los mensajes, diseñamos una nueva entidad CoAP, llamada CoAP relay. Esta entidad proporciona el procesamiento de mensajes mas sencillo, simplemente creando un túnel basado en CoAP para enviar el mensaje original. La última entidad diseñada es el CoAP stateless proxy, que tiene algunas características de las dos entidades anteriores. No mantiene ningún estado por smart object, ya que esta información se envía en los mensajes, y al mismo tiempo es capaz de analizar y procesar los mensajes para reducir los bytes enviados a través de la red. Estos tres intermediarios ayudan al smart object para unirse a la red de manera transparente. Esto significa que el smart object no está al tanto que el bootstrapping se está realizando a través de una entidad intermediaria hasta el final del proceso. Esta contribución se prueba en un LR-WPAN. Además del diseño y la definición de cada contribución, esta disertación también proporciona un análisis de rendimiento de cada uno de ellos. En primer lugar, mostramos que el el protocolo de bootstrapping tiene menos sobrecarga, gracias a un tamaño de mensaje menor que un protocolo de bootstrapping estándar actual para IoT PANA. Evaluamos el tiempo que toma realizar el bootstrapping en diferentes escenarios, con varios saltos entre los smart

objects y la entidad a cargo de la autenticación, así como también probamos diferentes índices de pérdida, para obtener una mejor comprensión del rendimiento que tiene cada solución. También evaluamos el porcentaje de procesos de bootstrapping que pueden finalizar con éxito en cada caso y el consumo de energía. Cuando la contribución se prueba en LR-WPAN, utilizamos el simulador de red de sensores para el sistema operativo Contiki llamado Cooja. La evaluación del rendimiento realizada en LPWAN utiliza una implementación real de LoRa. En conclusión, en esta disertación, diseñamos un servicio de bootstrapping construido en la parte superior de AAA, EAP y CoAP, para admitir una gran cantidad de dispositivos, cubriendo los requisitos mencionados para despliegues a gran escala. El servicio de bootstrapping define un protocolo de bootstrapping, en la terminología EAP, una EAP lower layer, que se construye sobre CoAP. CoAP se usa para lograr una sobrecarga baja, reduciendo el número de bytes utilizados para transportar mensajes EAP, en la comunicación entre el smart object y el Controlador, que es la parte de la red con limitaciones. Usando CoAP reutilizamos el código en los objetos inteligentes, ya que se espera estará presente en la mayoría de los dispositivos.

El servicio de bootstrapping ofrece una autenticación flexible y administración de claves mediante el uso de EAP. Esto permite la generación de material clave para ejecutar los protocolos de asociación de seguridad que se ajustan mejor a la tecnología de capa de enlace subyacente. Utilizando Infraestructuras AAA, proporciona soporte de federación de identidades, habilitando que dispositivos de diferentes organizaciones se desplieguen y operaren como parte del dominio de seguridad. AAA proporciona autorización flexible, utilizada generalmente para el acceso a la red, que puede ampliarse para proporcionar más capacidades de autorización de grano más fino. También proporciona capacidades de contabilidad que son útiles para rastrear el uso de los recursos de red para, por ejemplo, detectar el uso indebido. El servicio de bootstrapping, también proporciona soporte para redes de salto múltiple, mediante el diseño de tres intermediarios diferentes que ayudarán al smart object a unirse a la red, cuando no sea capaz para llegar al Controlador. Este servicio de bootstrapping se prueba en simulador en LR-WPAN y se compara con PANA que es el EAP lower layer estándar actual para el bootstrapping en IoT. Se rediseña y optimizada para LPWAN y proporciona a estas tecnologías las herramientas necesarias para proporcionar interoperabilidad, facilidad de gestión y federación de identidades entre otros servicios.

Pensamos que el servicio de bootstrapping propuesto en esta disertación proporciona una valiosa alternativa en bootstrapping en el Internet de las cosas.

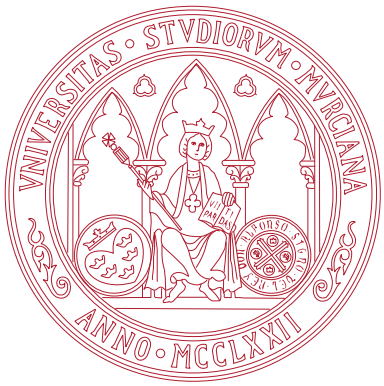
Agradecimientos

Quiero agradecer a todos los que han hecho posible la realización de esta tesis doctoral. En primer lugar y de manera especial a mis directores de tesis, Rafael Marín López y Antonio Skarmeta Gómez. Gracias por vuestro tiempo, vuestra fe y la inversión que habéis hecho para hacer este trabajo posible.

También agradezco a dos grandes equipos, como son el grupo de investigación ANTS de la Universidad de Murcia y el equipo de OdinS donde he podido finalizar la tesis.

No quiero olvidar a mi familia, mis padres y mis hermanas, quienes me han apoyado en estos años y a mi novia Belkys, quien no dudó de que pudiera conseguir este objetivo.

Finalmente, quiero dar las gracias a todos los que de una u otra manera han contribuido conmigo en trabajos de investigación, artículos, conferencias, IETF drafts, etc.



University of Murcia

Faculty of Computer Science

A CoAP-Based Bootstrapping Service for Large-Scale Internet-of-Things Networks

PHD THESIS

Author:
Dan García Carrillo

Thesis Advisors:
Prof. Dr. Rafael Marín López
CTO Odin Solutions Prof. Dr. Antonio F. Skarmeta Gómez

Murcia, July 2018

Abstract

According to the Oxford living Dictionary (on-line edition), the *Internet of Things* (IoT) refers to the interconnection of computing devices embedded in everyday objects via the Internet, so enabling them to transmit data. These embedded devices are generally called smart objects due to their capacity to interact with their environment. They may be sensors or actuators depending on whether they gather information or perform some physical action. Through smart objects the IoT enables advanced services to manage infrastructures such as smart homes, smart buildings, smart cities, etc. efficiently, or the productive processes of a company, such as maintaining and locating its inventory.

The IoT has the potential of having a positive impact in the economy and for society in general. Examples are applications that let people monitor their health and wellbeing, improving the productive process of a business, controlling the energy consumption of a population through a Smart grid, etc. All of this is possible thanks to smart objects being connected to the Internet.

Due to its potential, interest in this area is growing, not only in the research community, but also in companies and standardization organizations. Indeed, there is work done, as well as ongoing work, to provide IP connectivity to the myriad of devices expected to be part of the IoT.

Among the different areas of research and development related to the Internet of Things, there is one of particular interest, which is Security. The reason is that smart objects deal with sensitive information. In particular, there is a concern about providing the necessary means to secure the communications, not only form the point of view of the information that can be obtained, but also because of the possible repercussions of performing actions that will have a physical impact through actuators.

Securing IoT communications is challenging, since it involves smart objects with a set of characteristics and limitations such as the radio technologies used, with different constraints in *Central Processing Unit* (CPU) , memory, energy consumption and bandwidth. Protocols used in conventional Internet-connected devices may not be applicable, so the protocols used to secure the IoT they need to be adapted or redesigned, in some cases. There is agreement in that, in order to secure the communications in the IoT , we need certain

security processes such as *authentication* (to determine the device's identity), *authorization* (to determine what the device can do) and *key management* (providing key material to secure the communications). These security processes are used in several use cases, such as to access a service or establish a secure communication channel between two parties, and they are also part of what is called bootstrapping.

Bootstrapping is the process that sets the basis for a smart object to securely join a network and protect the communications so that it becomes a trustworthy party within a security domain. In general, a bootstrapping solution for IoT assumes that there is going to be a *Smart Object*, which intends to join a network formed by several smart objects, which conforms the constrained part of the network. The network is part of the security domain of a *Controller*, which is a device that is not as constrained as the smart objects. To perform the bootstrapping, both entities will communicate through a *bootstrapping protocol*, which implies the message exchange of the protocol with the objective of establishing an initial security association (SA) between the Smart Object and the Controller, which allows the Smart Object to become a trustworthy party in the security domain. Because the bootstrapping protocol runs on a constrained link, it needs to be as lightweight as possible.

Currently, there are bootstrapping protocols that are used in the context of IoT , but these generally assume that the devices are going to be managed under the domain of an organization with a single Controller, and do not take into account that the smart object could change Controller, moving to a different security domain, or that smart objects of other organizations could be installed in those deployments. We identify those deployments which are small to medium scale, where there is no need to use advanced management to bootstrap a large number of devices, such as the consideration of having devices from various organizations. Some other bootstrapping protocols are designed for a specific link-layer technology and do not account for the diversity of technologies in large-scale deployments, as in the case of a University Campus or a Smart City, with different buildings, green areas, car parks, etc. and with each facility, perhaps, belonging to a different administrative domain. In other cases, bootstrapping protocols have not been designed with the different constraints of IoT in mind. These are the reasons why we identify an important gap when it comes to bootstrapping, since current bootstrapping solutions do not generally account for large-scale deployments, and if they do, they are not suitable for constrained devices and networks.

There are certain requisites that we identify in a bootstrapping service for large-scale deployments. It should work *regardless of the link-layer*, aiming for a *lightweight* bootstrapping protocol. This could be achieved by trying to *reuse code* as much as possible and by reducing the number of bytes sent over the network. Due to the possibility of different link-layer technologies, the bootstrapping solution needs to provide *flexible key management*

to run the security association protocol that best fits the specific deployment. It also needs to account for the possibility of having devices from different organizations, to support *identity federation*. *Flexible authentication* is required to support different authentication mechanisms depending on the deployments (e.g., the smart object capacity and the restrictions in the link) and to adapt to company policies that may require the use of a specific authentication method. To provide information for the bootstrapping and the security association we need *flexible authorization* method that sends the authorization parameters (specific credentials, bootstrapping lifetime, etc.). Also, *accounting* is needed to track the use of network resources. Finally, we need to support different topologies, for which multi-hop networks need to be supported in the bootstrapping solution.

In this dissertation, we delve into the area of bootstrapping in the Internet of Things to provide a bootstrapping service for large-scale deployments that takes into account the previous requisites. To achieve this, we provide three main contributions in this thesis.

CoAP-EAP is a bootstrapping service built on 3 pillars: *Authentication Authorization Accounting* (AAA) infrastructures, *Extensible Authentication Protocol* (EAP) and the *Constrained Application Protocol* (CoAP). We present the architecture and the general flow of operation. CoAP-EAP offers a bootstrapping service that is independent of the link layer. It uses CoAP as a transport for EAP (implements an EAP lower layer in EAP terminology), so reducing the overhead in comparison with other EAP lower layers currently used in the IoT (such as *Protocol for Carrying Authentication for Network Access* (PANA)), and at the same time reuse code, using CoAP that is expected to be present in a large number of smart objects that provide services in the IoT. This contribution is tested in a *Low-Rate Wireless Personal Area Network* (LR-WPAN), as a representative of *Wireless Personal Area Network* (WPAN).

LO-CoAP-EAP is presented as a consequence of the relevance gained in recent years, by a new set of radio technologies, called *Low-Power Wide-Area Network* (LPWAN). This set of technologies offers long range communications, sending few bytes with a reduced energy consumption. LPWAN has high constraints in the link, which calls for a redesign of the bootstrapping service, that takes into consideration these constraints to provide a bootstrapping service that is independent of the link-layer technology. LO-CoAP-EAP provides a lightweight alternative that adds flexibility to the bootstrapping process in LPWAN , achieving a trade-off between performance and flexibility, as well as providing identity federation in LPWAN. We present the architecture and the general flow of operation, and we test LO-CoAP-EAP using the *Long Range* (LoRa) radio technology and a LoRaFabian network in a real deployment in Rennes, France. This contribution is also tested in a LR-WPAN to asses that the improvement can also be leveraged in this kind of network.

In the last main contribution, we consider the case of multi-hop networks where to perform the bootstrapping, a smart object may not be able to contact the entity in charge of the authentication (Controller) directly. In this case the Smart Object has to rely on another entity to aid in the bootstrapping process. This may be because the Smart Object does not have a routable IP until the bootstrapping is completed, or it is not able to reach the Controller by radio. To cover these possibilities, we define an intermediary entity, which is added to the architecture of the bootstrapping service. This entity is instantiated in three variants. A proxy, which is based on the CoAP proxy entity defined in the CoAP standard. This entity allows the analysis and modification of messages, as well as storing a state related to the message exchange. To have an entity that does not have to analyze the messages, we design a new CoAP entity, called CoAP relay. This entity provides the simplest processing of the message by simply creating a CoAP-based tunnel to send the original message. The last entity designed is the CoAP stateless proxy, which has some characteristics of the previous two entities. It does not keep any state per smart object, since this information is sent in the messages, and at the same time is able to analyze and process the messages to reduce the bytes sent over the network. These three intermediaries help the smart object to join the network in a transparent manner. This means that the Smart Object is not aware that the bootstrapping is being performed through an intermediary entity until the end of the process. This contribution is tested in a LR-WPAN.

In addition to the design and definition of each contribution, this dissertation also provides a performance analysis of each of them. In the first place, we show that the bootstrapping service and the bootstrapping protocol have less overhead, thanks to a lower message size than a current standard bootstrapping protocol for IoT PANA. We evaluate the time it takes to perform the bootstrapping in different scenarios, with several hops between the smart object and the entity in charge of the authentication, and we test different loss ratios, to gain a better understanding of the performance each solution has. We also evaluate the percentage of bootstrapping processes that are able to finish successfully in each case and the energy consumption. When the contribution is tested in LR-WPAN , we use Cooja —the wireless sensors network simulator for the Contiki operative system. The performance evaluation done in LPWAN uses a real LoRa deployment.

In conclusion, in this dissertation, we design a bootstrapping service built on top of AAA, EAP and CoAP , to support a large number of devices, covering the aforementioned requisites for large-scale deployments. The bootstrapping service defines a bootstrapping protocol in EAP terminology, an EAP lower layer, that is built on top of CoAP. CoAP is used to achieve a low overhead, by reducing the number of bytes used to transport EAP messages, in the communication between the Smart Object and the Controller, which is the constrained part of

the network. Using CoAP we reuse code in the smart objects, since it is expected to be present in most devices. The bootstrapping service offers flexible authentication and key management by using EAP. This allows the generation of key material to run the security association protocols that best fit the underlying link-layer technology. By using AAA infrastructures, the bootstrapping service provides identity federation support, enabling devices from different organizations to be deployed and to operate as part of the security domain. AAA provides flexible authorization, generally used for network access, that can be extended to provide more fine grained authorization capabilities. It also provides accounting capabilities that are useful to track the use of the network resources, for example to detect misuse. The bootstrapping service, also provides support for multi-hop networks, by designing three different intermediaries that will aid the smart object to join the network, when it is not able to reach the Controller.

This bootstrapping service is tested in a simulator in LR-WPAN and compared with PANA, which is the current standard EAP lower layer for bootstrapping in IoT. It is redesigned and optimized for LPWAN and provides these technologies with the necessary tools to provide interoperability, ease of management and identity federation among other services. We believe that the bootstrapping service proposed in this dissertation provides a valuable alternative to existing bootstrapping in the Internet of Things.

Acknowledgements

I want to thank all those who have made possible the realization of this doctoral thesis. First of all and in a special way to my thesis directors, Rafael Marín López and Antonio Skarmeta Gómez. Thank you for your time, your faith and the investment you have made to make this work possible.

I also thank two great teams, such as the ANTS research group at the University of Murcia and the OdinS team where I was able to finish the thesis.

I do not want to forget my family, my parents and my sisters, who have supported me in these years and my girlfriend Belkys, who did not doubt that I could achieve this goal.

Finally, I want to thank all those who in one way or another have contributed with me in research papers, articles, conferences, IETF drafts, etc.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

This opening chapter gives a brief contextualization of the Internet of Things. It focuses on the area of security, specifically bootstrapping, outlining the gaps and drawbacks that have motivated the work in this dissertation. After that, it describes the specific problem statement of this thesis, including the large-scale scenarios where current bootstrapping solutions do not provide a satisfactory solution. We then introduce our proposal to provide a bootstrapping service for large-scale IoT networks. After that, the main contributions and objectives of this thesis are described. Finally, the chapter details the structure of this document and lists the publication that have resulted from the research carried out.

1.1 The Internet of Things

The term "Internet of Things" was coined by Kevin Ashton in a presentation in 1999 [19]. The term reflected a way of improving the process of gathering information from the real world, where machines, instead of people, get the information. This is based on the premise that machines are more efficient and accurate than people in some tasks. He worked with *Massachusetts Institute of Technology* (MIT) on this concept and he realized that just as people have "sensors" all over their bodies to gather information about their environment, sensing technology could be used in conjunction with the Internet to potentially obtain a virtual nervous system. But the concept itself is nothing new, as he himself recognized, since the field of sensor networks had been studied for some time already.

The incipient Internet of Things started with projects such as connecting devices other than computers to the Internet. The first Internet *thing* was an IP–enabled toaster that could be turned on and off over the Internet, which was featured at an Internet conference in 1990 [120]. Other “things” became IP–enabled, a soda machine at the Carnegie Mellon University in the US [203] and a coffee pot at the University of Cambridge in the UK [155].

These playful experiments with a robust field of research and development into "smart object networking" [202] backed the creation of the foundation for today's Internet of Things.

There is a common understanding that the IoT intends to provide with Internet connectivity to a large number of devices, including devices with very high constraints. In fact, the Oxford dictionary defines *Internet of Things* as "*The interconnection via the Internet of computing devices embedded in everyday objects, enabling them to send and receive data*". Indeed, an important part of IoT networks is conceived to be formed by a great number of devices with constrained capabilities, called "smart objects" [113]. They are devices that are able understand and react to their environment, with IP-based networking connectivity [202]. As a consequence, it is estimated that in 2020 there will be more than 20 billion devices connected to the Internet [143].

The advances of the Internet of Things suppose an impact on technical, social, and economic areas. Consumer products, commercial goods, modes of transport, industrial processes and utility components such as sensors and other everyday objects are being combined with the Internet to provide new services through connectivity and powerful data analysis capabilities that "promise" to transform the way we work and live.

It is estimated that the IoT will have an economic impact of more than $11 trillion by 2025 [124]. Among the different fields of application we highlight *Smart Home*, to monitor and remotely manage home controllers and security systems; *Smart Cities*, to monitor public spaces and infrastructures; *Smart Buildings*, to manage and monitor building infrastructures such as lighting, heating, ventilation and air conditioning (HVAC), etc.; wearable technology to monitor human health; transport systems, where vehicles are monitored and provide information for more efficient transport systems, etc. the IoT is also leveraged by industry to improve their production process, where smart tags help to keep track of inventories, they can manage fleets of trolleys through the use of location services, generating efficient routes using live information, temperature control for sensitive cargo, etc.

The main drivers of the IoT vision are economic and technological advances (which enable this important growth). Some of these drivers are the low–cost of devices, pervasive network connectivity, especially through licensed and unlicensed wireless services and technology, making almost everything "connect-able". There is also an important investment in research, development, manufacturing, etc. delivering an ever increasing computing power at lower prices and at a lower power consumption. Cloud computing, providing networked computing resources to process, manages and stores data, allowing small and distributed devices to interact with powerful back-end analytic and control capabilities.

The IoT hosts diverse technologies, with physical and *Message Authentication Code* (MAC) layers such as Bluetooth and IEEE 802.15.4 for short-medium range communications,

and a relatively recent set of technologies, known as LPWAN, with long range communications (up to several kilometers) and a very low bandwidth among other restrictions. IEEE 802.15.4, which forms what are known as LR-WPAN, is used in scenarios such as *Smart Buildings*, while LPWAN are more suitable for large distances because of their long range capabilities, with low energy consumption, such as some *Smart Cities* applications, like lighting and parking management, and waste collection [14, 176].

However, managing the different technologies under the IoT is one of the current efforts; to homogenize the communications through the Internet Protocol (IP) that is becoming the dominant standard for networking. This provides a well–defined and widely implemented platform of software and protocols built on top of IP, which can be incorporated into a broad range of devices easily and inexpensively. In particular, what enables the great number of devices that are expected to be connected to the Internet is the IP protocol version 6 (IPv6 [45], which extends the number of IP addresses in comparison to the previous IPv4 version to "astronomical" values (2^{128}). In fact, one of the efforts following the "everything connected to the Internet" motto, is the adaptation of the IPv6 protocol for use in devices with constraints that do not allow running IPv6 as it is. This is part of the work of the *Internet Engineering Task Force* (IETF) *IPv6 over Low power Wireless Personal Area Networks* (6LoWPAN) Working Group, to make IEEE 802.15.4 devices IPv6-enabled, which operate independently of the link-layer. This effort can also be seen in Bluetooth, where recently the IETF released a Request For Comments (RFC) where it is specified how to provide IPv6 to Bluetooth [139]. In addition, LPWAN is the focus of the same effort from the recently formed IETF IPv6 over Low Power Wide-Area Networks (lpwan) Working Group that is working to provide these networks with IP connectivity. This tendency towards the homogenization of IoT communications can be seen as well at the application layer. The Constrained Application Protocol (CoAP) [190], of the IETF *Constrained RESTful Environments* (CoRE) working group, provides constrained devices with the capacity to offer their set of functions (e.g., sensing, acting, etc.) accessible as a web service, akin to the way we access the resources of a web server through the Hyper Text Transfer Protocol (HTTP) [60]. CoAP is established in LR-WAN and, currently, the LPWAN working group is working to adapt CoAP to the constraints of LPWAN networks [128].

Despite the advances and current efforts to realize the IoT and establish the basis of IP communications in these networks, it is not without its challenges. There are important issues that still need to be addressed. The involvement of massive scale deployment beyond traditional Internet-connected devices, brought by the IoT, presents a set of challenges related to security that has to be considered [149]. The security challenges attributed to the Internet [78] are still applicable to the IoT, with the added complexity of having massive deployments

of devices with different capabilities and a wide range of constraints or limitations (CPU, memory, energy, etc.) with different radio technologies that, overall, may limit the security solutions that are usually applicable to the Internet, having to adapt them to be usable by the smart objects. The subject of security is of great importance and we need to approach it in ways that consider the features and characteristics brought by the IoT, which is paramount to the successful development, adoption and coexistence with the Internet of Things.

1.2 Security in the Internet of Things

Security is a broad subject with many implications at different levels, such as physical security to avoid tampering with the devices or attacks against the communications. These are amongst the several security threats that can be found in the IoT [123]. especially, as mentioned by Pathak [149], the IoT opens new doors to attack due to the specific features of the devices forming IoT networks. They involve a massive deployment of devices, which can be done with collections of identical or similar devices. This means that if a vulnerability is found in one device, it can compromise the whole system, which is called a "domino effect". The maintenance and upgradeability of a large number of devices have to be considered; it is expected that smart devices will have a long service life and security mechanisms can become obsolete if they are not updated frequently. For this reason, there need to be protocols and measures in place to perform upgrades, which is not an easy task with such a large number of devices. Furthermore, there is no visibility of the internal operation of the IoT device, beyond some *Light-Emitting Diode*s (LEDs) that may be present in the device and it is not usual to find a basic screen showing some information about the device. This makes the maintenance or verification of devices difficult. Additionally, when these devices are deployed, securing them physically is a difficult task. Anyone with physical access to the device could compromise it. Overall, if any problem occurs, there is no quick reaction time, under an attack or any other problem. Related to security, interoperability and standards compliance allows the creation of robust systems as part of an agreement of several manufacturers.

Among the different areas where security is a concern, this dissertation focuses on the area of security in the communications. Security is a basic requirement to any communication system and, as such, security has to be among the various aspects to consider when designing an IoT system. Communication security is important because smart objects manage sensitive information. It may be personal information from end users such as a wearable device used for monitoring the health of a user. Sensitive information can be associated to company as well. A company could manage information regarding the location of their vehicles and the

personnel using them, as well as stock management, etc. Another reason why security is of great importance is because IoT systems may cause physical harm. They integrate sensors and actuators to interact with, and report about, the real world. If a group of sensors and actuators determine and regulate, for instance, the current temperature of sensitive cargo, not having security measures to assure the correct workings of the systems in place may incur in loss of revenue. Worse still, it might be a possible harm to human health, as it has been reported that medical devices such as insulin pumps, and pacemakers have vulnerabilities [116]. To prevent these we need to rely on security mechanisms like authentication, to verify the identity of a user or a smart object, and authorization to check the permissions before allowing any action by a user or smart object, as well as key distribution, to provide the necessary key material to protect the communications.

For this reason there are issues related to security that have to be taken into account, such as good design and deployment practices. Cost vs quality trade-offs should not compromise security. In this sense, secure standards should be the norm when designing new IoT devices and services, since they provide mechanisms and protocols agreed upon by several manufactures to ensure a consistent and inter-operable working systems. Furthermore, devices should be upgradeable when a new vulnerability is found. IoT systems that do not implement correct security measurements are subject to infection by malicious code (e.g., malware). They could also steal information not only pertaining to the basic operation of the device, but also code or algorithms that may be proprietary, so hurting the image and possibly the company's bottom line, wreaking havoc in the network if they are not authenticated or authorized.

Therefore, the Internet of Things undoubtedly presents a set of challenges in terms of security, due to the huge number of devices that are expected and their constrained capabilities in most of the cases.

There are several works [123, 170, 81, 87, 100, 109, 215] that define the challenges and features that are expected regarding security, which we summarize:

1. Authentication: In the IoT, smart objects must be able to clearly identify and authenticate other smart objects. This is related with the Identity management issue outlined earlier.

2. Authorization: We need to limit the use of resources to the ones the smart object is authorized to use and report on the attempts to use resources that the smart object is not entitled to use.

3. Key Management: Smart objects and other entities in IoT networks need to exchange cryptographic material to ensure confidentiality, integrity and authentication of the

data. Hence, they need lightweight key management systems to enable trust between different smart objects.

4. Confidentiality: Data confidentiality is paramount to avoid misuse of the information and acquisition by unintended parties. It is important to provide the tools to secure the information, in spite of the limitations the devices may have.

5. Integrity: We need to ensure the accuracy of the data, that it comes from the right and that the data is not tampered during transmission.

6. Objects safety and security: Physical safety of the smart objects is important, as they could be tampered with, stolen, etc.

7. Lightweight Security Solutions: Due to the limitations in the computational and power capabilities of the smart objects, these restrictions must be considered while designing and implementing security protocols in the IoT.

8. Heterogeneity: The network can be formed by the inter-connection of different types of smart objects with different capabilities, technologies, vendors or organizations. Because of this, protocols must be designed to work in all different smart objects as well as in different situations [216, 117, 168]. This is possible by providing a homogeneous level of communication through protocols independent of the link-layer.

9. Federated Architecture: A federated architecture can be used to overcome the heterogeneity authentication mechanisms [117], so easing the deployment, management of IoT systems.

Among the characteristics to make an IoT network communication system secure, there is a clear emphasis on four aspects that are key, as explained in works such as [149, 100]. These are *authentication, authorization, key management and data confidentiality* [149, 100]. These key aspects are related to the term bootstrapping [81, 69], which is the main focus of this thesis. Bootstrapping is a process that allows a smart object to join a network securely and become part of the security domain, performing its operation as a trustworthy entity within the domain. The problem of bootstrapping is not only a current topic of research, but there are also standardization organizations, such as the IETF, with working groups like 6TiSCH [95], *Authentication and Authorization for Constrained Environments* (ACE) [96], CoRE [94], etc. which are tackling the subject, the *Institute of Electrical and Electronics Engineers* (IEEE), *Internet Protocol for Smart Objects* (IPSO) [189], Zigbee [217], *Open Mobile Alliance* (OMA) [201], etc. which have devoted efforts towards the progress in this area in the Internet of Things, which we will review in Chapter 2.

1.3 Bootstrapping in the Internet of Things

The term "bootstrapping" is not new. It was coined in the early 1950s to refer to a load button that was used to initiate a bootstrap program or small program to start the operating system. In computer communications, one of the first protocols to use the term bootstrap was the *Bootstrap Protocol* (BOOTP) [40] to provide an IP address to a communicating host. This term has evolved, to incorporate security aspects. More recently, it has been referred to in the literature as the process by which a node gathers any necessary information (not only an IP address but also security parameters) to join a network or a service after an authentication and authorization process [135, 204] has been completed. Those security parameters can be, for example, available cipher-suites, shared-keys, certificates, service parameters, *etc*. Different services can initiate a bootstrapping process tailored to their specific needs. For example, *Mobile IPv6* (MIP6) considered as a service, requires dynamic parameters (material, home address IPv6 configuration, etc.) to be configured for its correct operation [148]. Another example is related to *Dynamic Host Configuration Protocol* (DHCP) authentication extension [49] where some cryptographic material that is required to protect DHCP is bootstrapped in a proposed solution using EAP transported over an extension of DHCP [154].

In the context of the Internet of Things, Garcia-Morchon *et. al* [70] describes the life-cycle of a smart object in the IoT as a 3-phase process, illustrated in Figure 1.1: *Bootstrapping, operation and maintenance*. After the device is manufactured, the device enters the *bootstrapping* phase, which entails the installation of the device, its commissioning, and the bootstrapping process. This includes providing the device with its identity and secret keys to set up the basis for its secure operation.

It starts when the smart object is turned on and seeks to join a security domain to perform its normal operation. The security domain is managed by an entity, which we will call the *Controller* throughout this thesis. The smart object and the Controller engage in the bootstrapping process, where the smart object is authenticated and authorized to join the security domain. After that, they establish a security association with the key material derived from the bootstrapping process. At this point, the smart object can perform its tasks during the *operation* phase, which we will refer to as post-bootstrapping in this dissertation. From time to time, a *maintenance* phase will be needed. Updating software, firmware, etc. after which, a re-bootstrapping is performed. Once the device is no longer usable, it can be decommissioned.

We can distinguish between a re-bootstrapping performed because the smart object has been updated, and can be considered as "new" to the Controller because there could be a change in the identifier of the device, and a re-bootstrapping performed because the associated

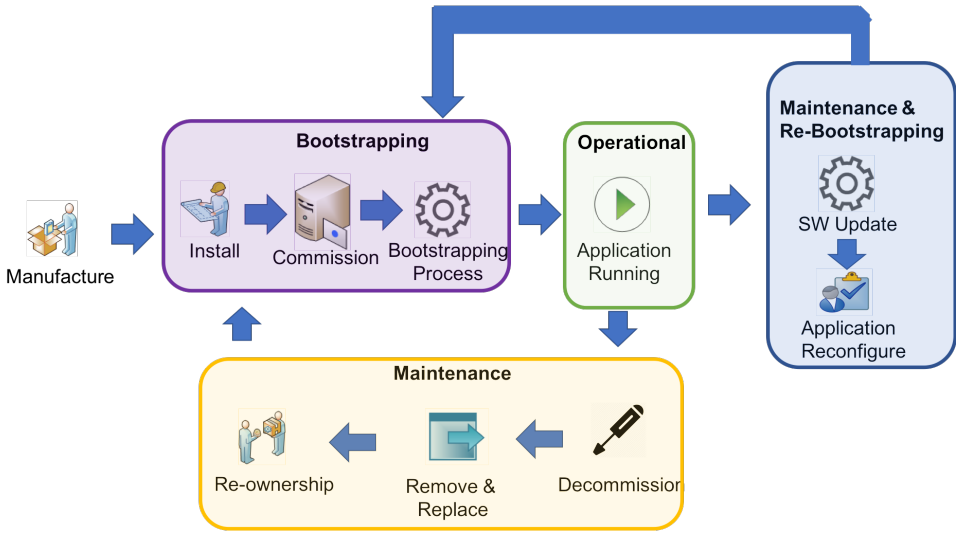


Fig. 1.1 Life-cycle of an Internet of Things (IoT) device

bootstrapping session is about to expire along with the key material and state related to it, which also need to be renewed.

As a consequence of the process, the smart object joins a security domain and *accounting* of the usage of the bootstrapping service can be carried out, collecting information about the operation of the smart object in the domain, which may be of interest to improve manageability. Thus, the bootstrapping process is especially important in the case of the IoT, where the configuration of the smart object is expected to be automated as much as possible, so improving the scalability of the IoT deployments more easily.

Thus, bootstrapping, as commented, rests on four pillars: authentication, authorization, accounting and key distribution, which are the security processes that we elaborate on next.

1. *Authentication*: The objective of this process is to identify the smart object. The identification can be made using different kinds of credentials: pre-shared keys, raw public keys or certificates installed during commissioning. Normally, the smart object contacts the Controller.

   The authentication process may be performed with different protocols such as *Authentication and Key Agreement* (AKA), authentication with *Pre-Shared Key* (PSK), *Transport Layer Security* (TLS), etc. The choice of protocol will depend strongly on the capacity of the smart objects, the limitations of the radio technology, and the policy of the organization that manages them.

2. *Authorization*: Once the smart object has been authenticated, the Controller has to verify the permissions the smart object has (e.g., joining the security domain, access

to the Internet, etc.), and gather the information regarding the bootstrapping such as an IP address, key material, the session time (which indicates when the device has to perform a re-bootstrapping), etc.

3. *Accounting*: The use of network resources by the smart object can be registered to track its activities, to manage and detect attacks or for billing.

4. *Key Distribution*: Once the authentication has been completed, and the smart object is successfully authorized to enter the security domain, key material is derived from the authentication process, which can be used to further derive key material to enable different security association protocols with the Controller to secure communications within the security domain.

Once the device has been bootstrapped, it has the basic key material, IP, authorization information, etc. and is able to start its normal operation, what we generally call post-bootstrapping. In post-bootstrapping, the device may need to establish security associations with different entities to protect the communications in order to accomplish its main objective (e.g. if it is a temperature sensor, send the current temperature to the pertinent server). In post-bootstrapping, different protocols can be used to acquire new credentials, such as OAuth from the IETF ACE working group.

1.3.1 Classification of bootstrapping solutions in the IoT

Figure 1.2 describes the different classifications that we make to distinguish the different kinds of deployments in terms of the bootstrapping process that can be expected in an IoT network: *small, medium and large scale*. We define small scale deployments scenarios as domains with one Controller where all devices and their credentials are configured in that Controller by the administrator of the domain. Medium scale deployments will have several domains with one or more Controllers managing each domain, but under the same organization. Large scale scenarios add the deployment of devices from different branches or campuses of the same organization, each with a different administrative domain, as well as the possibility of having devices from different organizations, adding a level of complexity to the security management and bootstrapping process.

Let us take an example. Let us imagine that the University of Murcia (UMU) deploys sensors and actuators to monitor a floor of the Computer Science Faculty. To that effect, a Controller is placed on the 1^{st} floor, for instance, to manage the different sensors that are deployed on that floor. In this scenario, the management could be quite simple, by putting all the information and credentials of the sensors that can be set up in the Controller to carry out

the bootstrapping. This illustrates the *small-scale* scenario. Now, let us imagine that we do not intend to manage only the Computer Sciences faculty building, but every building in the campus, each with one or more Controllers. In this case, the administrator would have to set up the credentials from each device to their assigned Controller. This task, while manageable, starts to add complexity to the deployment and management of the scenario. For instance, if we were to change a sensor from one floor to another, we would have to configure its credentials into the new Controller and remove them from the previous one. Here we are instantiating a *medium-scale* scenario.

Adding more complexity to the previous deployment, the UMU has different buildings, green areas, car parks, etc. The University of Murcia has 5 campuses, each campus with one ore more buildings to manage. If the administrator were to manage all sensors in this case, he would be facing an important undertaking, as the task of setting up the credentials to each Controller, the maintenance or change of location of the devices will be very difficult. This instance represents a *large-scale* deployment where each campus might have different administrators and they have to collaborate to set up the complex scenario. In addition to the *large-scale* deployment, we may find that other organizations, such as law enforcement or firefighters may deploy sensors and actuators to monitor and manage certain alert situations. In this context, the smart objects that belong to the University of Murcia will not have a problem with authentication as they are deployed in their own campus, and UMU manages their identities, permissions, and the conditions of their use of the network resources. A different case is found when we deal with the devices deployed by law enforcement or firefighters. To be able to manage those devices, there are two options: 1) The University of Murcia takes charge of managing external devices, with the accompanying issues, such as manually registering each device, new or replaced devices, setting up the permissions according to the agreed upon service agreement between the two entities, etc. 2) The organization that owns the devices (e.g., police, or firefighters) takes care of authenticating and authorizing its own devices, and sends the information to the UMU through bilateral agreements between both organizations. Thus the Controller/s in UMU interact with smart objects of external organizations and relay the authentication and authorization process to the home organization of these smart objects to verify if the devices are authenticated and authorized to be deployed and operate in UMU.

Therefore, we can see how in large-scale deployments, identity federation, representing the second option, becomes relevant and useful. Figure 1.2, also shows a generic federation architecture and illustrates how devices from organization X (e.g., POLICE) have their smart objects commissioned by their *Identity Provider* (IdP) X, but the device is deployed in domain A (UMU). The smart objects of organization X have to interact with their own organization's

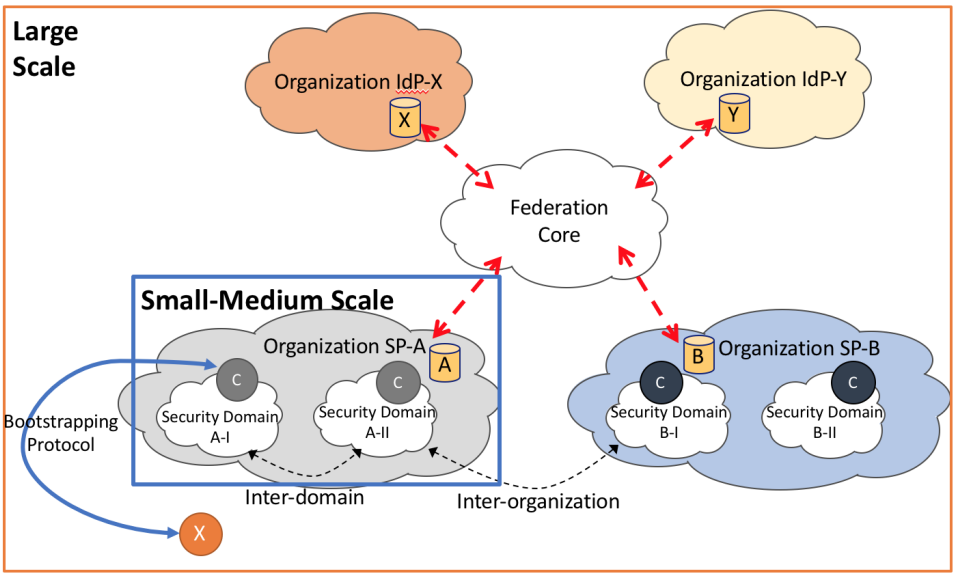


Fig. 1.2 Generic federation architecture of a bootstrapping service

IdP X, relieving the organization where the device is deployed and which provides the service, the Service Provider (SP) A, of storing or managing the credentials of devices that are not from the organization. The process would be as follows: The smart object from organization X is deployed in organization A. To bootstrap the device, it contacts the Controller of its security domain in organization A. Being from a different organization, the Controller relays the authentication process to the IdP of organization X, which authenticates the device and informs the Controller of organization A if the device is successfully authenticated and authorized to be deployed there, in which case it receives the necessary information to manage the bootstrapping of the smart object (e.g., bootstrapping expiration time, key material, parameters of the use of the services such as network access, etc.) with which the Controller and the smart object can establish a security association and set the smart object as a trustworthy entity in the Controller's domain.

In addition to the already mentioned uses cases pertaining to large-scale scenarios, there is another area of applicability, where a smart object changes the domain where it is deployed. Changing to another domain managed by a different Controller is referred to as *mobility*. Figure 1.2 shows this possibility with the inter-domain and inter-organization arrows. In addition to the mobility within the same organization (inter-domain), a device could also move to a different domain and require the performance of a bootstrapping there. If we perform the bootstrapping in a way that is independent of the technology, smart objects could be deployed in any domain.

1.3.2 Requisites for a large-scale bootstrapping solution

Providing a bootstrapping solution covering different possibilities that may happen in large scale deployments, also covering small-medium scale deployments, supposes a level of complexity that calls for a solution that eases the management process, that is able to bootstrap those devices regardless of the domain where they are located and the underlying link-layer technology. To accomplish this we need a solution that centralizes the management to make it more efficient, managing the credentials in one place instead of having to manage a number *n* of Controllers to set up the credentials in each one, making the solution *scalable* by using protocols and infrastructures designed to manage a large number of devices.

Additionally, to perform the bootstrapping of a smart object in a *large-scale* scenario we need to understand the different types of deployments. It is very likely that there will be deployed devices with different capabilities (CPU, memory, etc.). Thus, to suit the needs of the more constrained devices we need to provide *lightweight protocols* that do not impose an overhead that some of the devices are not capable of handling. Related to this, we need to consider the constraints of the devices themselves, as they may have limited memory. In this sense, we should *reuse code* whenever possible. We can also expect different kinds of radio technologies, with different restrictions in the link, such as bandwidth, duty cycle, etc. Examples of this are LPWAN, LR-WPAN, etc. When providing a solution to bootstrap devices with different link-layer technologies, we need a solution that is *link-layer independent*. Otherwise, we would have to design a solution tailored for each one with the added drawbacks of lack of interoperability. Due to the different capabilities of the devices and the constraints of the link-layer technology and possible restrictions by the organization that owns the smart objects, we need to offer *flexibility in the authentication method*, choosing the method that best fits each case. Additionally, each organization managing its smart object may impose specific authentication methods. Hence we need to achieve a trade-off between a lightweight solution and flexibility. Also, depending on the characteristics of the smart objects, we find that the different Security Association Protocols (SAP) are used to protect the communications once the smart object has key material from the bootstrapping process. For instance, the IEEE 802.15.9 standard describes several options (e.g., IKEv2, *Host Identity Protocol Diet EXchange* (HIP-DEX), etc.), and, additionally, there are the IETF's proposals for the IoT such as *Datagram Transport Layer Security* (DTLS) or *Object Security for Constrained RESTful Environments* (OSCORE). DTLS is is heavy in very constrained networks [207], for which alternatives like OSCORE [185] might be used. This calls for a *flexible key management* with a framework that is independent of the technology to provide the key material needed to run the particular security association. We need, also, to provide a *robust and flexible authorization* mechanism, adapted to the devices and multi-domain

scenarios, that is centered in the performance and extensible. Considering also the possibility of having devices that belong to different organizations, we need to provide support to *identity federation*, as well as track the use of the different resources by the devices performing the consequent *accounting* process. Finally, we will mention that to account for different network topologies in the IoT, we have to consider the possibility of performing the bootstrapping in a multi-hop network, for the case where the device is deployed in a mesh network and does not have a routable IP to reach the Controller. This adds complexity to the scenario, since we are not assuming always a direct (1-hop) communication between the smart object that joins the network and the Controller, and we need to devise a solution to provide bootstrapping in these networks too. Taking these aspects into consideration, we can summarize a set of requirements that a solution for bootstrapping in large-scale deployments should meet:

(R1) *Link-layer independent*: As new radio technologies become part of the IoT, we need protocols that are independent of the link layer to maintain interoperability, ease management and avoid modifying the bootstrapping solution or designing new ones.

(R2) *Lightweight protocols*: The diversity of devices and link-layer technologies with varied constraints requires protocols that impose small overhead on the device and the use of the network.

(R3) *Identity federation*: It is possible that, in a particular deployment, devices from different organizations have to co-exist or even cooperate to accomplish their goal. These devices are registered in different domains from where they are deployed. Hence, they managed by different IdPs that belong to different organizations and must be authenticated and authorized.

(R4) *Flexible key management*: Each technology or standard may favor different protocols to secure their communications after the bootstrapping, and a bootstrapping solution that is generic needs to cater to the needs of the different security association protocols that may be used.

(R5) *Flexible authentication*: Different capabilities in hardware or software in smart objects can impose different restrictions on the authentication method to be used, or the organization policy may establish specific authentication methods to be used by their devices. Hence, support for different authentication methods is required.

(R6) *Accounting*: Tracking the use of the resources by the different devices can serve different purposes, such as management, analysis of the activities of a security domain, detecting attacks or for billing companies using the resources.

(R7) *Robust authorization*: The process of authorization needs to provide well established authorization mechanisms for the process of bootstrapping.

(R8) *Support for multi-hop networks*: To cover the different types of networks in the IoT we need to support multi-hop networks, where the smart object that is deployed in the domain may not reach the Controller of the domain directly, but though another smart object or other entity that is acting as an intermediary, which will need further consideration when designing a bootstrapping solution.

(R9) *Reuse code*: Given the constraints of some of the devices, we need to consider the reuse of existing code whenever possible, achieving the bootstrapping using a protocol that is already present in the devices, rather than using a protocol solely for this purpose.

(R10) *LPWAN support*: Given the inclusion of this set of technologies with high constraints in the IoT, we also need to consider them in a general bootstrapping solution.

(R11) *Scalable*: The bootstrapping solution has to be scalable, otherwise it is not a large-scale solution.

1.4 Current alternatives in bootstrapping in the IoT

Bearing in mind the requirements gathered in Section 1.3.2, we analyze the related work in the area of bootstrapping in the IoT. There are several ways of performing bootstrapping in the IoT. The set of solutions can be classified into two big groups: small-medium scale and large scale.

Small to medium scale solutions' current work focuses mostly on LR-WPAN networks. These are scenarios, like Smart buildings or homes, where the number of devices to be managed may allow for local configuration of all the devices deployed and where there is no need for large scale infrastructures. Works on bootstrapping in small-medium scale scenarios, [68, 24, 112, 158, 189, 4] usually rely on pre-shared key material and on simply using the pre-shared key to run a security association protocol (SAP) between the smart object and the Controller. Just running the SA is what is considered as bootstrapping. For instance, in a home automation scenario, only DTLS may be used with pre-installed key material, enabling a secure channel between two communicating entities. In turn, these solutions can be separated among those specially designed for a particular link-layer (link-layer alternatives); those that assume the use of DTLS (DTLS-based alternatives), and others using HIP-DEX (HIP-DEX-based alternatives) or using CoAP (CoAP-based alternatives):

- *Link Layer alternatives:* The IEEE 802.15.9 [5] defines a message exchange framework based on information elements (IE) as a transport method for *Key Management Protocol* (KMP) datagrams and guidelines for the use of some existing KMP s with IEEE Std 802.15.4, such as HIP-DEX, minimal IKEv2, PANA, Dragonfly and 802.1X, to establish a security association, transporting them over the link-layer instead of *User Datagram Protocol* (UDP), as opposed to solutions like DTLS, HIP-DEX, etc. Another alternative, the encapsulation of EAP over IEEE 802, is defined in the *EAP over LAN* (EAPoL) specification [1]. Hernández-Ramos *et al.* [85] design an optimization over EAPoL for bootstrapping resource constrained devices, called Slim EAPoL (SEAPOL) performing the functionality using 3 bits instead of the 6 bytes used in EAPoL. The problem with this solution is that it is not complete, since they only uses EAP for authentication but does not discuses how key derivation is done to secure the MAC layer. In general, link-layer protocols are designed to improve the performance of the bootstrapping process by transporting the protocols at this layer, offering an optimized alternative. However, it means they are only applicable to those link layers.

- *DTLS-based alternatives:* DTLS alternatives such as Bergmann *et al.* [24], Garcia-Morchon, et. al [68], the Open Mobile Alliance (OMA) [158], are based on establishing a security association, usually based on pre-shared keys for authentication and authorization management. The issues we find with DTLS is that these solutions are heavyweight for constraint networks deployments [207], lack accounting and federation support, and are not manageable in large scale deployments, since the configuration has to be done by each Controller. DTLS is not sufficient to perform flexible key management to secure different link-layer technologies, and the use of other protocols is needed, as in the case of [68].

- *HIP-DEX-based alternatives:* HIP-DEX [132] is a lightweight alternative to *Host Identity Protocol* (HIP) [133], and HIP-DEX alternatives such as Meca *et al.* [126], Garcia-Morchon, et. al [68], and Colin [144], are based on establishing a security association based on Elliptic Curve Cryptography (ECC) Diffie-Hellman, although there is a variant that uses pre-shared keys (HIP-PSK) [68]. HIP-DEX is used for authentication and network access, and may support federation [140]. Although HIP-DEX is independent of the link layer and is designed to work in constrained devices, public-key cryptography using certificates that can be too expensive for constrained devices, it lacks flexible authentication, key management and accounting, and offers only single-hop bootstrapping, which not suitable for bootstrapping in multi-hop networks.

- *CoAP based solutions.* There are some solutions for bootstrapping that use CoAP. Kang *et al.* [103], which designs a bootstrapping solution using pre-shared keys to derive key material to protect the communications. Korhonen [112] proposes using the Generic Bootstrapping Architecture (GBA) defined by *3rd Generation Partnership Project* (3GPP) to perform the bootstrapping of smart objects, using pre-shared keys and removing AAA for the sake of simplicity. The aforementioned solutions lack flexibility in the authentication and identity federation.

Among the *large scale solutions*, we find that the new set of technologies known as LPWAN does consider the use of AAA [59]. In IEEE 802.15.4 they usually use PANA and in some cases AAA.

- *LPWAN solutions* secure their communications at link-layer, using pre-shared keys [192, 210, 91]. LoRaWAN [197], for example, defines its own protocol to join the network, called Joining Procedure. Where AAA infrastructures are incorporated, as in the case of Garcia-Carrillo *et al.* [65, 102] with LoRaWAN, flexible authorization and federation is provided, but the authentication method does not change, and neither does the dependence on the link layer, which requires to design a solution for each LPWAN technology.

- *EST-based solutions*: Enrollment over Secure Transport (EST) [37] is a protocol for bootstrapping certificate and the associated Certification Authority (CA) certificates over TLS and HTTP. The IETF *Autonomic Networking Integrated Model and Approach* (ANIMA) working group uses *Enrollment over Secure Transport* (EST) for a solution for automated *Bootstrapping Remote Secure Key Infrastructures* (BRSKI) [152], using certificates that are conceived for large scale, but it is not considered for constrained devices. Other solutions, such as EST over secure CoAP [47] propose an adaptation of EST for constrained devices that work on top of DTLS. This has the consequent limitations, as recognized in the document, that some devices will lack the resources to handle large payloads managed in est-coaps. These solutions are not lightweight, lack flexible authentication or multi-hop support.

- *PANA-based solutions* such as [144, 178, 42, 174, 217] use the Extensible Authentication Protocol (EAP) with PANA, and in some cases the interaction with AAA infrastructures. PANA is a current standard protocol for bootstrapping in the IoT. It is used in Zigbee IP [217] and is usually proposed for bootstrapping in the IoT. Das and Ohba [42] propose the use of an EAP based framework, using PANA as a transport for EAP, for bootstrapping and CoAP for provisioning credentials to DTLS-PSK and

Table 1.1 Characteristics of the current bootstrapping solutions

Solution	*(R1) Link-Layer Independent*	*(R2) Lightweight*	*(R3) Federation Support*	*(R4) Flexible Key Management*	*(R5) Flexible Authentication*	*(R6) Accounting*	*(R7) Flexible and Robust Authorization*	*(R8) Multi-hop topologies*	*(R9) Code Reuse*	*(R10) LPWAN*	*(R11) Scalability*
LoRaWAN with AAA [65, 102]	NO	YES	YES	NO	NO	YES	YES	NO	NA	YES	YES
IEEE 802.15.9 [5]	NO	YES[2]	YES[2]	YES	YES	YES[2]	YES[2]	YES	NA	NO	YES[2]
DTLS [24, 68, 158, 207]	YES	NO	NO	NO	YES	NO	NO	NO[1]	YES	NO	NO
HIP-DEX [126, 144, 68, 140]	YES	YES	NO[1]	YES	NO	NO[1]	NO[1]	NO	NO	NO	NO[1]
EST[152, 47]	YES	NO	YES	YES	NO	NO	NO[1]	NO[1]	NO	NO	YES[2]
PANA[144, 178, 42, 174, 217]	YES	NO	YES[2]	YES	YES[2]	YES[2]	YES[2]	YES	NO	NO	YES[2]
CoAP[103, 112]	YES	YES	NO	NO	NO	NO	NO	YES	YES	NO	NO

[1] The feature can be added even though is not specified in the standard
[2] The feature is present, but some specifications may limit the usability of the feature

the PSK mode of IKEv2. The issue here is that PANA was not designed with the constraints of the IoT in mind, and it imposes an important overhead in the constrained link, as we will analyze during this thesis.

As summarized in Table 1.1, of the different solutions for bootstrapping in the IoT, PANA fits best the list of requirements gathered in Section 1.3.2. The problem with PANA, taking as an example its use in Zigbee IP, is that Zigbee IP fixes the EAP method to EAP-TLS. This limitation is hard to justify, because the purpose of EAP is precisely to offer flexibility in the authentication method. If a fixed authentication method is going to be used, other alternatives, such as DTLS would also be valid. Furthermore, Zigbee IP do not use AAA infrastructures, limiting large-scale and multi-domain deployments, but it could be integrated. Up to this point, PANA fulfills, potentially, all the criteria we are looking for in a bootstrapping solution for the IoT. The problem is that PANA was not designed for the constraints of the IoT, as we will see throughout this work, due to their big message size and the number of messages exchanged during the bootstrapping.

1.5 A bootstrapping service for large-scale IoT networks

In this thesis, we design a new bootstrapping service to fulfill all the requisites defined in Section 1.3.2, given that current alternatives lack one or more or the characteristics for bootstrapping in a large-scale deployment. The bootstrapping service needs to provide a lightweight bootstrapping protocol, reusing code whenever possible, flexible authentication, supporting different authentication methods depending on the devices characteristics, with robust authorization and to provide key management to run different security association protocols. Accounting for the use of the resources, for which the smart objects are authorized is also important. Tracking the use of these resources can be used to adapt the service to the agreement between the organizations, which would require support for identity federation. Finally, we consider the possibility of multi-hop networks.

1.5.1 Providing the federation substrate: AAA

In order to support identity federation in the context of bootstrapping and network access is usually done using AAA infrastructures. The AAA Framework [44] provides support for the three basic security services in network deployments: authentication (to determine who the end user is), authorization (to determine under what conditions an end user is granted access to the network resource), and accounting (to register the resource consumed by the end user). Thus, it is consistent with at least two of the required processes, authentication and authorization, involved during the bootstrapping. In this sense, we foresee the key importance of AAA-based infrastructures. Indeed, they are widely used nowadays to large-scale deployment for two reasons: first, they are robust infrastructures for managing authentication, authorization and accounting for the activities of the smart objects. AAA in conjunction with the Extensible Authentication Protocol (EAP) [10] provides a secure framework for flexible authentication, authorization and key distribution [88, 12]. Furthermore, AAA authorization mechanisms can be extended with other protocols such as *Security Assertion Markup Language* (SAML) to provide more fine-grained authorization information [89]. Second, AAA is widely used to manage a great number of device connections, thereby supporting large scale deployments. In fact, AAA infrastructures based on the protocol Diameter [35] are commonly used in 3G-5G networks to control the access of millions of users. Another example is the *educational roaming network* (eduroam) [212], which is a world-wide federation for WiFi connectivity across campuses and research and educational organizations around the world, which supports thousands of users. Eduroam deploys EAP and an AAA infrastructure based on the Remote Authentication Dial In User Service (RADIUS) [167], which provides the identity federation substrate.

Before choosing this alternative, there are other identity federations nowadays. OAuth 2.0 [77] can perform delegated authorization, giving the user who requests access to some resource owned by another entity the necessary credentials to access those resources. OpenID [161] can perform authentication and is used to improve the experience of web users, having to use only one ID (their OpenID) to be authenticated for multiple services. *OpenID Connect* (OIDC) (OIDC) [171] is an authentication layer on top of OAuth 2.0. It allows clients to verify the identity of a user based on the authentication performed by an authorization server in an inter-operable and REST-like manner. SAML [169] is a protocol that provides assertions to exchange authentication and authorization data. OAuth, is only for authorization, so we would not be able to support authentication and accounting natively. OpenID, in contrast, is used for authentication. In this case we would be missing authorization and accounting. OpenID connect and SAML are used for both authentication and authorization and could be feasible. In terms of authorization we need mechanisms that are more oriented to network

Table 1.2 Identity Federation comparison

Technology	Authentication	Authorization	Accounting	Scalability
OpenID	YES	NO	NO	YES
OAuth2.0	NO	YES	NO	YES
OpenID Connect	YES	YES	NO*	YES
SAML	YES	YES	NO	YES
AAA	YES	YES	YES	YES

access and everything related to it, defining attributes and information such as IP filters, quality of service, key distribution for network access, etc. since it is closely related to bootstrapping because it is the first time the device is going to access the network. This authorization should be extensible in order to add flexibility. Additionally, we still have the need for accounting and AAA is widely used as a mainstay in the provision of the aforementioned set of security services that allow secure network access through different wireless technologies. In fact, we are not the only ones to see the potential of AAA in the IoT, as there is currently work towards their integration in 5G networks [9].

Table 1.2 shows a summary of the aforementioned Identity Federation technologies. The only technology that supports the three A's are the ones implementing the AAA model. Of the other technologies, OIDC is the most comprehensive, since it supports authentication and authorization. As reviewed in [30], for OIDC to provide accounting, it needs to support that feature separately, which is why its marked with an asterisk in the table. We fall in favor of AAA, besides having accounting as part of their model, because it supports network access authentication natively. Additionally, the other Identity Federation technologies focus on access to resources and services once the user can access the network, in what we call the post-bootstrapping phase.

1.5.2 Providing flexible Authentication: EAP

Regarding the requisite of flexible authentication, AAA is used in conjunction with the Extensible Authentication Protocol (EAP) [10], which, as we mentioned, provides a secure framework for flexible authentication, authorization and key distribution. EAP is a protocol for authentication that provides the necessary tools to perform the bootstrapping and provides a greater level of flexibility and granularity than other authentication protocols. It can be used with AAA for a federated deployment and it provides key material as a consequence of the authentication that can be used to further secure the communications. Indeed, EAP allows different types of authentication mechanisms (e.g., based on symmetric keys, digital certificates, etc.) called EAP methods without change EAP itself. For example, EAP-PSK

[25] is an EAP method based on the use of a pre-shared key (PSK) to provide a lightweight authentication mechanism. Other examples of EAP methods, such as EAP-AKA and EAP-TLS, can be found in [156]. Furthermore, the use of novel EAP methods for the IoT is considered in works like Granlund *et. al* [73] that define EAP-Swift, an EAP method for authentication and key derivation for constrained devices, and Aura et. al [20] which works on EAP-NOOB, an EAP method for nimble out-of-band (OOB) authentication and key derivation, specifically for IoT devices. EAP is, therefore, a strong candidate to perform authentication in the IoT due to the flexibility it brings to the authentication. To exemplify the features brought by EAP instead of relying on pre-shared key material, that is the solution adopted by a great number of current alternatives, we can see how *Wireless Fidelity* (WIFI) works with two different options: Enterprise Extensible Authentication Protocol (EAP), and Pre-Shared Key (PSK). WiFi Protected Access 2 (WPA2) with PSK, is the most used in home deployments. The use of *Wi-Fi Protected Access* (WPA); EAP, PSK or any WPA Enterprise (i.e. EAP) implementation is not intended (per se) to improve cryptographic strength (although EAP gives the flexibility to choose the authentication method that best suits the needs of each case) of a wireless network, but to provide additional characteristics, such as granular control over who or what connects to the network. With the EAP options under WPA-Enterprise each user and device can have its own credentials, and this increases security and auditing. EAP has a well defined key management framework (EAP *Key Management Framework* (KMF) [12]) that defines a key hierarchy and provides a framework for the transport and use of the parameters and keying material that is generated by EAP authentication algorithms, known as EAP methods. Thus, EAP is used in conjunction with AAA to provide flexible authentication and key management, which are part of the requirements for large-scale bootstrapping in the IoT. To use EAP, we need an EAP lower layer, a protocol that fulfills the requirements described in [10] to transport its messages between the EAP peer, which is the entity that has to be authenticated (i.e., the smart object), and the EAP authenticator, that steers the authentication process (i.e., the Controller). As we will see in Chapter 2, there are several EAP lower layers, some of which are considered in the IoT. One of those is PANA, an EAP lower layer used in bootstrapping protocols such as Zigbee IP and IEEE 802.15.9 in the context of the IoT. Interestingly, however its design is previous to the IoT 's becoming mainstream, and it was not designed with the constraints of the IoT in mind. This is why we must consider an alternative EAP lower layer to exchange the EAP messages in the constraint link and take into account the constrained capabilities of the smart objects. A question that arises is, why are we not transporting EAP at the link layer, as 802.1X does with EAPoL, transporting EAP over Ethernet frames. While there are works in this direction [85, 5], we considered that using link-layer dependent technologies

has several issues, namely: 1) the variety of radio technologies used in the Internet of Things, which is growing, each with its own specific link-layers. 2) Each technology defines its own security measures, how to protect the communications, and their bootstrapping process (if any). 3) This heterogeneity brings interoperability and management issues, having to account for each of the technologies supported. These issues, brought by the diversity of radio technologies, are what call for a link-layer independent solution. Otherwise, each technology would have to define its own adaptations at its link layer.

We aim, instead, at a solution that is independent of the link layer and that can be used in highly constrained links, considering that there will be a trade-off between performance and interoperability gained by dissociating the bootstrapping protocol from the technology used. This is possible by designing a bootstrapping service that is based on a protocol specifically designed for IoT networks. This is why we will compare throught this dissertation our bootstrapping service only with PANA. Because, following the definition of bootstrapping in this work, the only bootstrapping protocols used in IoT are PANA and 802.1X. From the two, 802.1X is link layer dependant with the limitation for single-hop authentication, not applicable to multi-hop topologies. PANA, on the contrary, is independent of the link-layer technology and is able to operate in mulit-hop topologies.

1.5.3 Providing a bootstrapping service based on CoAP

To design a lightweight protocol to perform the bootstrapping we need to use a protocol that is designed considering the constraints of the IoT. Since the EAP lower layer transports EAP messages in the constrained link, we are interested in achieving a low overhead with a bootstrapping service independent of the link-layer. This is the Constrained Application Protocol (CoAP) [190]. CoAP is a standard protocol used in constrained devices and networks, such as the IoT, that works on top of UDP, providing a link-layer independent protocol. It is a web transfer protocol based on the REST [163] model, especially designed for constrained devices with shorter message lengths and less demand on resources. If CoAP were used as bootstrapping protocol, we would have an alternative that is lightweight in comparison with the current standard for bootstrapping in the IoT (PANA), as we will see throughout this dissertation.

For the aforementioned reasons, we use CoAP to design a lightweight bootstrapping service for large-scale IoT networks. This bootstrapping service is based on three pillars: AAA, EAP, CoAP and is called CoAP-EAP. The CoAP-EAP bootstrapping service, will use CoAP as bootstrapping protocol. For this reason the bootstrapping service defines a new EAP lower layer based on CoAP to transport EAP messages. Up to this point we have covered almost all the requirements.

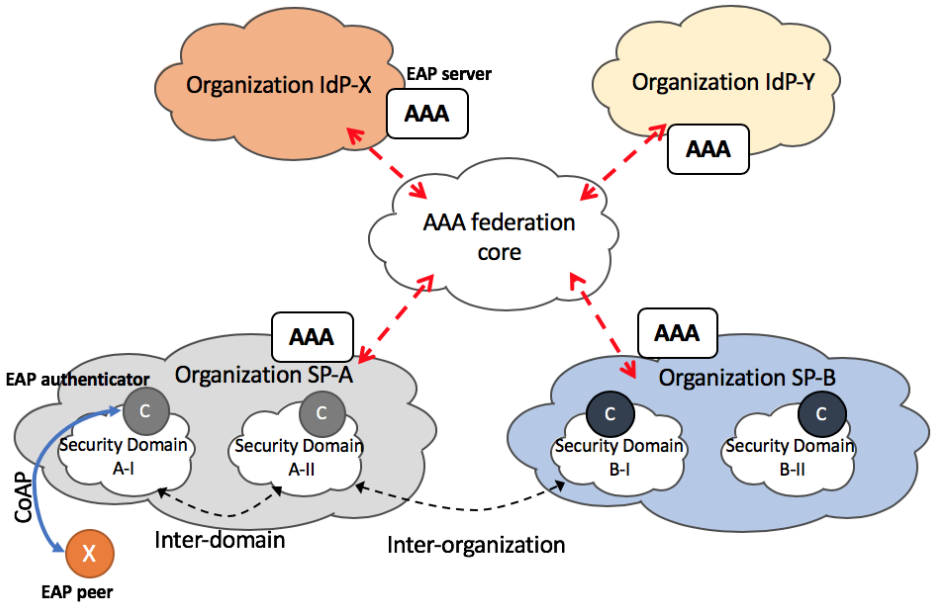


Fig. 1.3 Bootstrapping service for large scale based on AAA federation

Now that we have a complete picture of the bootstrapping service, we can see (Figure 1.3) an instantiation of the generic federation architecture shown previously, using in this case the protocols that conform our contribution. In this case, the identity federation substrate is provided by the AAA infrastructure. The different organizations AAA servers are interconnected through the AAA federation CoRE. When a new device seeks to join a security domain (e.g., *A-I*), it will reach the Controller of that domain (EAP authenticator) using the bootstrapping protocol, in this case CoAP, to begin the bootstrapping process, and perhaps through an intermediary. The Controller in the Service Provider of the organization (*SP-A*) will communicate with the smart object Identity Provider *IdP-X* (EAP server) to perform the authentication process. Once the authentication is finished, the Controller receives the confirmation from the IdP, if the devices is successfully authenticated. The IdP sends the Controller the necessary information related to the authorization of the device and its integration in the security domain, such as the IP, lifetime of the generated credentials, etc. using the AAA protocol, at which point the device enters the security domain of the Controller (*A-I*) as a trustworthy entity.

1.6 Main contributions and objectives

Throughout the chapter we have seen that bootstrapping is a keystone process to secure the communications in the IoT. We saw that large-scale deployments are expected to be com-

monplace in a global IoT and we need a bootstrapping solution that covers the requirements elicited in Section 1.3.2, which no current bootstrapping solution covers them all.

After reviewing several technologies, we have concluded that to provide this bootstrapping service we are going to use AAA to provide the federation substrate. We will use EAP in conjunction with AAA to provide flexible authentication and key management. Because we need a lightweight EAP lower layer (a protocol to transport EAP) in the constrained link, we use CoAP. Hence, the main contribution of this dissertation is based on three pillars: AAA, EAP and CoAP to provide a lightweight bootstrapping service for the IoT, which we call CoAP-EAP.

In this PhD we have three main contributions related to the bootstrapping service for large-scale IoT networks:

- **CoAP-EAP**: The first contribution in this dissertation is the design of the CoAP-based bootstrapping service for the IoT. We define its architecture and the general flow of operation, specifying how the Controller and the Smart Object are able to establish a security association using the key derived from the EAP authentication. Additionally, we specify how this key material can be used to derive additional key material to be used to run a security association protocol (exemplified with DTLS) to secure the communications between the Smart Object and the Controller. We test CoAP-EAP in Cooja, a networks simulator for *Wireless Sensor Network* (WSN) using the Contiki Operative System, and we compare it to PANA. We see an improvement in general, in the time, percentage of successful bootstrapping, energy and memory use. This contribution covers requisites *R1, R2, R3, R4, R5, R6, R7, R9* and *R11*.

- **Low Overhead CoAP-EAP (LO-CoAP-EAP)**: During the research for the first contribution the set of technologies known as LPWAN started to gain relevance, and we were contacted by researchers at the IMT Atlantique (former Telecom Bretagne) because of their interest in the bootstrapping service and its applicability in this new set of technologies called LPWAN. Even though the bootstrapping service already designed (CoAP-EAP) is lightweight, we have to account for the strong constraints LPWAN imposes to be able to provide LPWAN with a suitable bootstrapping service. We had to optimize the current solution, sending only the essential to the bootstrapping service. This led to the design of LO-CoAP-EAP (Low Overhead CoAP-EAP), which provided the same CoAP-based bootstrapping service, with a reduced overhead, sending fewer bytes over the network. LO-CoAP-EAP was tested on a real world LoRa deployment in Rennes, France. We find that LR-WPAN deployments can also benefit from LO-CoAP-EAP, which is also tested in Cooja to confirm that it also provides

an improved performance in comparison with the original design. This contribution covers the same requisites as the previous contribution, with a special emphasis on *R10* due to the restrictions of LPWAN.

- **Bootstrapping in multi-hop IoT networks**: The last contribution is focused on providing bootstrapping in multi-hop networks where the Controller is not reachable by the smart object. For this, we define an entity (called intermediary), which we will instantiate through CoAP intermediary entities, to aid in the bootstrapping process. We design and compare three intermediary entities: A CoAP proxy, which is able to maintain a state related to the authentication and modify the messages; a CoAP relay, which is not present in the CoAP standard, and does not keep any state related to the authentication and does not modify the messages; and a CoAP stateless proxy, which is a hybrid of the previous two, able to modify the messages, but not maintaining any state related to the exchange. We expand the architecture of the bootstrapping service with the intermediary entity, and define the message flow with each intermediary as well as explaining how the Controller manages each intermediary. We design the intermediaries to be transparent to the Smart Object, in the sense that it is not aware that it is talking with an intermediary until the authentication is finished, and the Controller sends the authorization information to the Smart Object, which lets it know it was talking through an intermediary.

  We test the three intermediaries in Contiki O.S. on a network simulator (called Cooja). We compare the performance of each intermediary with the PANA relay, and we find that the CoAP intermediaries impose less overhead than the PANA intermediary. This contribution covers requisite *R8*.

Taking into account the contributions and the requirements that we have to cover, this thesis has four main objectives:

(O1) Design a bootstrapping service that is adapted to the constraints of the Internet of Things, using CoAP as a transport for EAP, and AAA infrastructures

(O2) Account for the severe constraints of not only LR-WPAN but also LPWAN, adapting the bootstrapping service to fit the constraints of LPWAN as well.

(O3) Support bootstrapping in multi-hop topologies.

(O4) Validate the solutions designed by means of prototype implementations, evaluating their functionality, feasibility and performance.

1.7 Related Publications

The research work carried out for this PhD has been published - or is under revision- in several peer-reviewed international journals.

1.7.1 Journals

- Garcia-Carrillo, D. and Marin-Lopez, R. (2016). Lightweight CoAP-based bootstrapping service for the internet of things. Sensors, 16(3):358. [66]
- Garcia-Carrillo, D., Marin-Lopez, R., Kandasamy, A., and Pelov, A. (2017). A CoAP-based network access authentication service for low-power wide area networks:Lo-coap-eap. Sensors, 17(11). [67]
- Multi-hop bootstrapping with EAP through CoAP intermediaries for the IoT ***(to be accepted)***. IEEE INTERNET OF THINGS JOURNAL.
- Hernandez-Ramos, J. L., Moreno, M. V., Bernabe, J. B., Carrillo, D. G., and Skarmeta, A. F. (2015). Safir: Secure access framework for iot-enabled services on smart buildings. Journal of Computer and System Sciences, 81(8):1452 – 1463. [84]

1.7.2 Conferences

- Bohli, J. M., Skarmeta, A., Moreno, M. V., García, D., and Langendörfer, P. (2015). Smartie project: Secure the IoT data management for smart cities. In 2015 International Conference on Recent Advances in Internet of Things (Riot), pages 1–6. [31]
- Hernandez-Ramos, J. L., Carrillo, D. G., Marín-López, R., and Skarmeta, A. F. (2015). Dynamic security credentials PANA-based provisioning for the IoT [83]

1.7.3 Book Chapters

- Jose L. Hernandez-Ramos, Dan Garcia-Carrillo, A. S. F. G. L. C. J.-M. B. M. B. (2016). Smartie: a secure platform for smart cities and the IoT. In Benjamin Aziz, Alvaro Arenas, B. C., editor, Engineering Secure Internet of Things Systems, chapter 4, pages 266–290. Institution of Engineering and Technology. [101]

1.7.4 Internet Drafts

- Lopez, R. and Garcia, D. (2017). EAP-based Authentication Service for CoAP. Internet-Draft draft-I-D.draft-marin-ace-wg-coap-eap, Internet Engineering Task Force. Work in Progress. [122]
- Garcia, D., Lopez, R., Kandasamy, A., and Pelov, A. (2016b). LoRaWAN Authentication in RADIUS. Internet-Draft draft-I-D.draft-garcia-radext-radius-lorawan, Internet Engineering Task Force. Work in Progress. [65]
- Kandasamy, A., Lopez, R., Garcia, D., and Pelov, A. (2016). LoRaWAN Authentication in Diameter. Internet-Draft draft-I-D.draft-garcia-dime-diameter-lorawan, Internet Engineering Task Force. Work in Progress. [102]
- Garcia, D., Garcia, S. N. M., and Lopez, R. (2016a). Application Layer Security for CoAP using the (D)TLS Record Layer. Internet-Draft draft-I-D.draft-garcia-core-app-layer-sec-with-dtls-record, Internet Engineering Task Force. Work in Progress. [64]
- Sarikaya, B., Sethi, M., and Garcia, D. (2018). Secure IoT Bootstrapping: A Survey. Internet- Draft draft-sarikaya-t2trg-sbootstrapping-04, Internet Engineering Task Force. Work in Progress. [179]

1.8 Outline

In chapter 2, we review the Background and State of the art related to bootstrapping. In Chapter 3, we present the main contribution of this work, the CoAP-EAP bootstrapping service for large-scale deployments. We describe the architecture and the bootstrapping protocol. We implement a proof of concept that is tested in the Cooja network simulator of the Contiki Operative System and CoAP-EAP is compared with PANA.

In Chapter 4, we present the adaptation of CoAP-EAP to LPWAN networks, called LO-CoAP-EAP. We define its architecture, the protocol flow, as well as the proof of concept implementation that is tested in the Cooja network simulator, and in a real LoRa deployment in Rennes, France. LO-CoAP-EAP is compared with CoAP-EAP as well as with PANA.

In Chapter 5, we present the extension of the CoAP-EAP architecture to support multi-hop topologies. We define its architecture, the protocol flow, as well as the proof of concept implementation that is tested in the Cooja network simulator, and it is compared with the PANA relay element. Finally, in Chapter 6 we offer the conclusions and future work.

Chapter 2

Background and State of the Art

The research presented in this dissertation is built upon various standard technologies and protocols. This chapter provides a brief description of the most relevant bootstrapping technologies. Since the bootstrapping service designed in this thesis is independent of the link-layer, and is intended to be used in different scenarios and devices, with different capabilities, we review the types of devices considered in the IoT, as well as the radio technologies used in the IoT. Specifically, Section 2.1 describes the different categories of smart objects, and the subset of smart objects for which we intend to provide a bootstrapping solution. Section 2.2 reviews several radio technologies used in IoT that are currently the focus of research in one or more areas, such as adding IPv6 support, supporting new topologies, developing security frameworks, etc. Since our work focuses on providing a link layer independent bootstrapping service, we review the most relevant technologies. Section 2.3 provides the background in the technologies used to design the bootstrapping service. Section 2.4 describes the work related to bootstrapping in the IoT and we review the work on bootstrapping in standardization organizations.

2.1 IoT Devices: Types, classes, categories

The IoT concept is intended to host a myriad devices with different characteristics and capabilities, depending on how they are powered (if they are limited by battery life), their computational power, amount of memory, the radio technology used, etc. In the context of this thesis, we will use the classification system in RFC7228 [33], which establishes some basic terminology for constrained-node networks, and classifies constrained devices in 3 classes, as shown in Table 2.1.

RFC7228 describes 3 classes of devices. The first class (C0) are devices that are not generally capable of sustaining an Internet connection with the basic underlying security

Table 2.1 Classification of IoT devices according to RFC7228

Class Name	Data Size (e.g., RAM)	Code Size (e.g., ROM)
Class 0	«10 KiB	«100 KiB
Class 1	∼10KiB	∼ 100 KiB
Class 2	∼ 50 KiB	∼ 250 KiB

protocols. They will usually communicate through other devices acting as proxies, gateways or servers. For this reason they are outside the scope of this thesis. Class 1, are constrained in memory and processing power. They will not support a typical full protocol stack (HTTP/TLS/TCP/IP) nor the use XML-based data representations, but they are capable of using specific protocols stacks for constrained devices, such as CoAP (see Section 2.3.3) over UDP, using IPv6 adaptations such as 6LoWPAN. Class 1 devices can communicate with other nodes through the Internet without relying on other devices. Finally, Class 2 devices are not as constrained and do not need to be restricted to protocols stacks for constrained devices. However, they can still benefit from the use of specific protocols stacks for constrained devices due to the memory and energy savings, and from the interoperability gained with other Class 1 devices. Constrained devices with capabilities superior to class 2 devices are not defined in RFC7228.

For the bootstrapping service proposed in this thesis we aim for devices that can maintain an Internet connection and have the processing power to establish secure channels (e.g., DTLS, OSCORE, etc.). To do this, we focus on Class 1 and 2 devices and any other device with capabilities superior to Class 2 that will benefit from a specific protocol stack for energy saving and interoperability purposes.

2.2 Radio Technologies in the IoT

There are a variety of radio technologies used in the Internet of Things. The most widely associated with the IoT are the Wireless Personal Area Network (WPAN) radio technologies, such as Bluetooth and IEEE 802.15.4, which provide low to medium range coverage, and are used in wearables and Smart Buildings.

There is a more recent set of radio technologies considered to contribute to the Internet of Things, known as LPWAN, that provide long range communications (up to several kilometers), with low energy consumption at the cost of sending payloads of few bytes.

2.2.1 Wireless Personal Area Network (WPAN)

IEEE 802.15 Wireless Personal Area Network (WPAN) is a set of technologies that provide a short-medium range communications. For example, IEEE 802.15.1, which is know as Bluetooth, in versions 1.1 and 1.2. The following versions are handled by the Bluetooth *Special Interest Group* (SIG). The latest version known as *Bluetooth Low Energy* (BLE) is one of the contestants in the IoT race. It offers considerable bandwidth (compared to LR-WPAN), enabling most of the wearables and gadgets known to date (mobile phones, smart watches, etc.). One of the handicaps of Bluetooth is that, until recently (RFC7668), there was no standard support for IPv6, which is one of the main drivers of the IoT. Moreover, the current standard does not provide multi-hop network support, only star topology. In contrast, IEEE 802.15.4,LR-WPAN, support mesh network topologies and IPv6 through 6LoWPAN [188].

2.2.2 Low-Power Wide-Area Networks (LPWAN)

There is relatively new set of radio technologies that support large distance communications (up to several kilometers) at the cost of a very reduced bandwidth. Applications that fit these specifications can be car park sensors, water meters, or smart garbage collection, smart agriculture, etc., minimizing infrastructure needed to manage a great number of devices.

Some of the technologies are LoRa [197], Sigfox[192], Weightless[91], *Developers' Alliance for Standards Harmonization of ISO 18000-7* (DASH) [210], NB-IoT [159], WISUN [23], etc. They can operate in licensed or license-exempt bands to provide connectivity great number of battery-powered devices. However, differences between the technologies make them incompatible in some cases. Here are some characteristics of these technologies.

- Very small frame payload, as low as 8 bytes.
- Very low bandwidth, most LPWAN technologies offer a throughput between 50 bit/s to 250 kbit/s.
- Depending on the technology, very limited message rate (e.g. between 0.1 message/minute and 1 message/minute) due to regional regulations that limit the duty cycle (e.g. from 0.1% to 10%) in some *Industrial, Scientific and Medical* (ISM) bands.
- High packet loss rate, which may be the result of bad transmission conditions between nodes.
- Variable *Maximum Transmission Unit* (MTU) for a link depending on the L2 modulation used.

- Currently some technologies lacking L2 fragmentation capabilities.
- Highly asymmetric and, in some cases, unidirectional links.
- Ultra dense networks with thousands to tens of thousands of nodes.
- Typically, star topology networks.
- Different modulations and radio channels within the same technology.
- Sleepy nodes to preserve energy.

With the goal of homogenizing the communications, the use of IPv6 is being considered in these networks applying specific compression mechanisms. This work is performed by the IPv6 over Low Power Wide-Area Networks (lpwan) IETF working group.

Regarding security, each technology has different methods to provide security for the communications. Bootstrapping and key management is not directly considered by every technology, because some solutions are not publicly accessible. Some LPWAN technologies just pre-install some key material that is shared with the server with which the node communicates, and when the device is turned on it sends the information (protected) to the server. Another issue is that not every technology cyphers the messages as in the case of Sigfox. Bootstrapping and key management is considered in LoRaWAN, Wi-SUN Alliance Field Area Network (FAN) and NB- IoT.

2.3 Background to the protocols used in the bootstrapping service

In this section we review several protocols and standards, as part of the background for this thesis. We look at the AAA framework, the Extensible Authentication Protocol (EAP) and the Constrained Application Protocol (CoAP) because these are the pillars of our bootstrapping service.

2.3.1 Authentication, Authorization and Accounting (AAA) Framework

The AAA Framework [44] provides support for the three basic security services in network deployments: *authentication* (to determine who the end user is), *authorization* (to determine under what conditions an end user is granted access to the network resource), and *accounting* (to register the resources consumed by the end user). Thus, it is consistent with at least two of

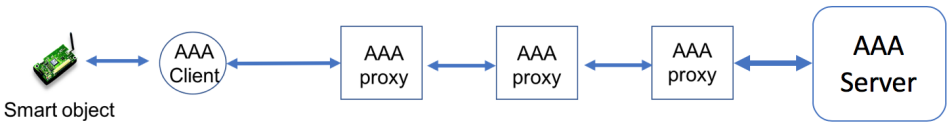


Fig. 2.1 AAA deployment with several AAA proxies between the AAA client and the AAA server

the required processes, authentication and authorization, involved during the bootstrapping. Accounting is in itself important, since it allows us to keep track of the activity within the network and can be used to detect the different activities within the network, adjust the parameters according to the service agreement, etc.

The AAA framework defines a model consisting of: an *End User* (EU) desiring access to a specific network service; an IdP which has registered the end user's identity and long-term credentials (e.g., a certificate or pre-shared key); and a *Service Provider* (SP) operating and controlling the access to the network service. These entities, in the case of the bootstrapping service, would be the smart object, Authentication Server and Controller, respectively. In a *non-federated case*, the IdP and SP belong to the same organization (*IdP 's organization*). In *federated cases*, where some bilateral agreements are assumed among different domains joining the federation, the IdP and SP belong to different organizations that operate on different AAA servers: *the IdP 's AAA server* and *the SP's AAA server*. Both AAA servers are capable of exchanging authentication, authorization information and accounting data.

Additionally, the SP also operates the entity that intermediates between the smart object and the AAA infrastructure to carry out the authentication and authorization processes. This entity is generally called *Network Access Server (NAS)*. Thus, the simplest AAA infrastructure consists of a *Network Access Server* (NAS) directly connected to a AAA server by a AAA protocol. Nevertheless, several AAA servers can be deployed between the NAS and the AAA server for scalability reasons or federated access support. AAA intermediaries can be relay or proxies. The relay only forwards the information, while the proxy is able to apply internal policies to take decisions about the messages. An example of this scenario is given in Figure 2.1, where the AAA client communicates (the SP of the organization where the smart object is deployed) with the AAA server of the smart object (the IdP of the smart object's organization) through several AAA proxies.

Either way, the NAS, AAA proxies, or relays, and AAA servers that constitute the AAA infrastructure exchange information by using AAA protocols. Today, the most commonly deployed AAA protocols are RADIUS [167] and Diameter [57]. Although the latter is the most complete, in terms of the functionality provided by its design, *Remote Authentication*

Dial-In User Service (RADIUS) is still one of the most widely deployed protocols within existing AAA infrastructures [121].

The *Remote Authentication Dial In User Service* (RADIUS) [167, 166] is a client-server protocol where a NAS acts as a RADIUS client transporting messages on top of UDP. The RADIUS client sends AAA information in a RADIUS message request to the RADIUS server, which answers with a RADIUS response. There can be two RADIUS servers for different purposes: the authentication and authorization server [167], which keeps a database with registered users, and the accounting server [166], which collects data from the service usage (data sent and received, connection time, *etc.*).

In contrast, Diameter [57] is an evolution to solve some of the RADIUS limitations in terms of scalability and security. It uses a reliable transport like *Transmission Control Protocol* (TCP) or SCTP [199]. For the security part, it uses either IPsec [107] or TLS [48] and provides a message format allowing a longer message size and a higher number of concurrent connections [135]. Thus, it is better adapted for a large number of end users. Which is why it is used in 3G, 4G and considered in 5G [142].

2.3.2 The Extensible Authentication Protocol (EAP)

The *Extensible Authentication Protocol* (EAP) [10] is an IETF protocol that allows different types of authentication mechanisms (e.g., based on symmetric keys, digital certificates, *etc.*) named *EAP methods*. For example, EAP-PSK [25] is an EAP method based on the use of a pre-shared key (PSK) to provide a lightweight authentication mechanism. Other EAP methods are EAP-AKA[17] and EAP-TLS [193]. More examples of EAP methods can be found in [41].

EAP is a lock-step protocol, which supports only a single packet, request or response, in flight. Each request message (*EAP Request*) is answered with a response (*EAP Response*). The number of exchanges will depend on the EAP method selected. Every EAP method runs between the *EAP peer*, and the *EAP server* through an *EAP authenticator*. From a security standpoint, the EAP authenticator acts as a mere EAP packet forwarder.

To perform an EAP authentication, the EAP authenticator usually starts the process by requesting the EAP peer's identity through an EAP Request/Identity message. The EAP peer answers with an EAP Response/Identity with its identity. The identity follows the *Network Access Identifier* (NAI) format [46] (e.g., *smartobject@domain*). It contains a smart object's identity information (*i.e.*, *smartobject*) separated with an @ and the domain (*i.e.*, *domain*) it belongs to. With this information, the EAP server will select the EAP method to be performed. The EAP method execution involves several EAP Request/Response exchanges between the EAP server and the EAP peer.

There are two deployment models of deployment for EAP. On the one hand, the *standalone EAP authenticator model*, where the EAP server is co-located in the same device as the EAP authenticator. This may be appropriate in deployments with a small number of objects. In this case, there is no AAA infrastructure in the back-end. On the other hand, there is the *pass-through EAP authenticator* model, which is the most scalable configuration. In this model, the EAP server and the EAP authenticator are implemented in separate nodes. Specifically, the EAP server is centralized on an AAA server located in the domain of the Identity Provider of the EAP peer and gives service to several EAP authenticators. Here, the communication between the EAP server and the pass-through EAP authenticator is performed using an AAA protocol. In both cases, a protocol referred to as the *EAP lower-layer* is used to transport the EAP packets between the EAP peer and the EAP authenticator. To the EAP peer, the model used is transparent, following the principle of mode independence. This means that to the EAP peer, the EAP conversation between the EAP peer and the server is unaffected, whether they are using pass-through mode or not. Figure 2.2 shows a mapping between the entities defined in the AAA model and EAP pass-through model. Each layer in the EAP processes a part of the EAP message in each entity, as described in [10]. As observed, the EAP method is processed in the EAP peer and the EAP server in the IdP 's AAA server.

The mode Independence of EAP mentioned previously serves to introduce what are called EAP invariants. These are characteristics that hold true for every EAP implementation. The 4 EAP invariants are: mode independence, media independence, method independence and cipher suite independence. *Mode independence*, as commented before, refers to the fact that the EAP peer is unaffected if the mode is standalone or pass-through. *Media independence*, specifies that all EAP methods will work as long as the EAP lower layer meets the specified requirements in the standard (we will review these in Chapter 3). *Method Independence* specifies that, even though the EAP authenticator might not implement an EAP method used by the peer, by enabling pass-through, authenticators can support any method that is implemented by both, the peer and server. This makes the inter-operation between the peer and authenticator possible, as long as the peer and server support a suitable EAP method. *Ciphersuite Independence* is required to support media independence. This invariant requires that exported keying material be large enough (with sufficient entropy) to handle any ciphersuite.

According to the RFC4017, there is a requirement for the EAP methods intended to be used in Wireless *Local Area Network* (LAN) s. It specifies that EAP methods need to generate symmetric key material, of 128-bits of effective key strength. It must provide mutual authentication support, be resistant to dictionary and man-in-the-middle attacks and

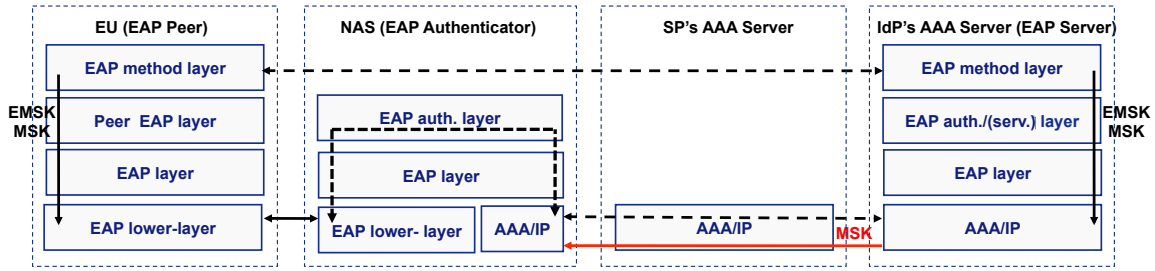


Fig. 2.2 Extensible Authentication Protocol (EAP) pass-through model.

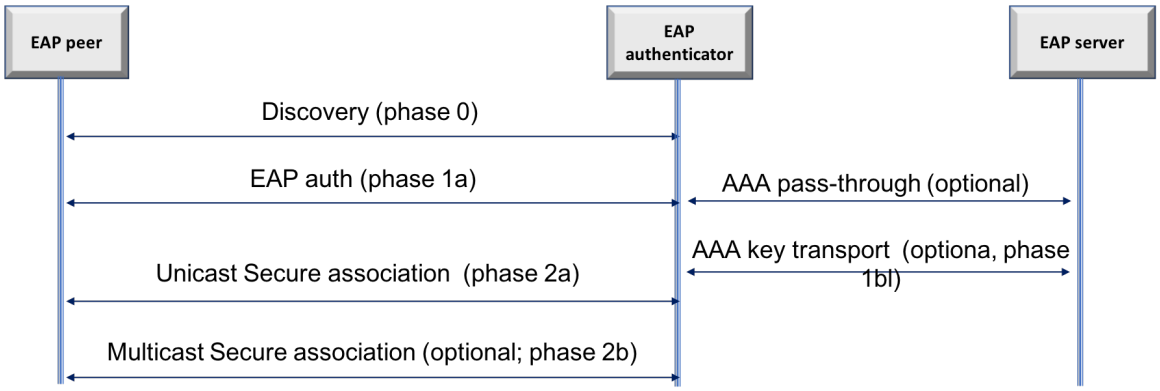


Fig. 2.3 Phases of the EAP Key Management Framework

protect ciphersuite negotiation. This rules out some EAP methods that do not provide the aforementioned requisites [198], highlighting the importance of key derivation and mutual authentication. To understand how this feature works, we review the EAP Key Management Framework and the EAP Key hierarchy.

The EAP Key Management Framework (EAP KMF)

The EAP KMF defines the EAP key hierarchy and provides a framework to transport and use key material and parameters generated by EAP methods. Where EAP key derivation is supported, the process is divided in in three phases, as is illustrated in Figure 2.3.

Phases 0 and 2 are managed by the EAP lower-layer protocol. Phase 1b is typically managed by a AAA protocol. The discovery (phase 0) the EAP peer (smart object) locates the authenticator (Controller). The discovery process may be automatic or manual, after which phase 1 (the authentication) starts. After the authentication is completed, if the EAP method that is used supports key derivation, EAP keying material is derived in both the peer and the EAP server. Another step (phase 1b) is needed in deployments with a backend authentication server, in order transport keying material from the backend authentication server to the authenticator. To oblige the principle of mode independence, in the case of a

backend authentication server being present, all the key material needed by the EAP lower layer is transported from the EAP server to the authenticator. After the authentication is completed successfully, the process of joining the network is completed when a security association is run between the EAP peer and the EAP authenticator. This Secure Association Protocol exchange (phase 2) between the EAP peer and authenticator is performed to be able to manage the creation and deletion of a unicast (phase 2a), and, optionally, a multicast (phase 2b), security association between both entities.

Following the EAP key management framework, we associate *Phase 1* to bootstrapping and *Phase 2* to post-bootstrapping. Phase 1 is consistent with the definition of bootstrapping given in Chapter 1, where the device is authenticated and authorized to enter the security domain of a Controller, and key material is derived to protect the communications between the Smart Object and the Controller. Phase 2 is also consistent with post-bootstrapping, because in this phase the Smart Object and the Controller can establish a Security Association (SA) to protect their communications and derive more key material for different layers of the network stack (e.g, the link layer).

The EAP Key Hierarchy

Certain EAP methods are able to generate keying material [156]. Specifically, according to the *EAP Key Management Framework* (EAP KMF) [12]. The cryptographic material that is derived by the EAP KMF after a successful authentication is composed of four symmetric keys.

- The *Master Session Key* (MSK) is a key of a minimum length of 64 octets. It is used as the main key to establish a security association between the peer and the authenticator. The *Master Session Key* (MSK) is sent by the AAA server to the authenticator and generated by the peer itself after the EAP authentication. To improve security in the distribution process of the MSK key, the EAP KMF has defined a security mechanism called channel binding that ensures that the parameters about the network service supplied by the authenticator, both the peer and EAP server are the same. These parameters depend on a particular access and are supplied through the lower layer and include data such as the MAC address of the authenticator and the peer.

- The *Extended Master Session Key* (EMSK), like the MSK, is cryptographic material exported by the EAP method and shared between the peer and the server. The *Extended Master Session Key* (EMSK) is at least 64 octets in length. Unlike the MSK, the EMSK should not be provided to any entity outside the peer or the server, so it is never transported to any other external entity. However, both entities can maintain and use

the EMSK to derive new keys. Note, however, that the original EAP KMF does not define any specific use for the EMSK. A general scheme defined for using EMSK can be found in [173], where it is specified that the use of EMSK should be for the sole purpose of deriving root keys and also specific mechanisms to avoid conflicts between root keys are given.

- The *Transient EAP Keys* (TEKs) are session keys used to protect the EAP conversation. The *Transient EAP Keys* (TEK) s are internal to the EAP method and are never exported to another entity. These keys are created during an EAP conversation and discarded once the authentication process finishes.

- The *Transient Session Keys* (TSKs) are the result of executing a security association protocol between the peer and the authenticator, and are derived using the MSK key as the root key. *Transient Session Keys* (TSK) s are used to protect the data traffic using the set of algorithms (cryptographic suite) negotiated between the peer and the authenticator through the security association protocol. this *Security Association Protocol* (SAP) is typically dependent on the underlying technology.

In particular, the MSK key is exported to the EAP lower-layer in the case of the peer, and to the AAA protocol stack in the case of the AAA server, where the EAP server is assumed to be located. Additionally, an EAP method exports another set of parameters that may be useful for certain applications. These parameters are the *Session-ID*, the *Peer-ID* and the *Server-ID*. The *Session-ID* uniquely identifies an EAP authentication process between the peer and the server. This session value is composed of the type of the EAP method executed concatenated by a unique identifier called Method-Id. The *Peer-Id* identifies the EAP peer involved in an EAP authentication session and conversely, the *Server-ID* identifies the EAP server.

2.3.3 The Constrained Application Protocol (CoAP)

Bormann *et al.* [32] discuss the possibility of smart objects being susceptible to offering resources or services represented by a *Uniform Resource Identifier* (URI). This has been solved in non-constrained deployments with HTTP. For constrained deployments, the *Constrained Application Protocol* (CoAP) [190] has been proposed as a web transfer protocol based on the REST [163] model in IoT. The main reason is that CoAP has been specially designed for constrained devices. CoAP provides a short message length that is less demanding in terms of parsing complexity. In fact, *Hypertext Transfer Protocol* (HTTP) might be too weighty for some types of smart objects. As analyzed in [32], HTTP poses a considerable

implementation burden that exceeds the capabilities of small devices. Indeed, IoT devices are constrained in memory, energy and computational resources and are expected to be deployed in constrained networks, where increasing the message length by a few bytes of information might result in fragmentation in the links [33]. A constrained application protocol as CoAP, with short message length and low computational requirements, therefore alleviates these problems. For all these reasons, although CoAP has certain similarities with HTTP, they are not compatible.

CoAP is designed to work on top of UDP, although, it is worth noting that the transport of CoAP messages in IEEE 802.15.4e frames is currently being considered [208]. CoAP has been designed with several features, of which we highlight: (1) low overhead and low parsing complexity; and (2) support for the discovery of CoAP resources and services. On the one hand, low overhead mainly refers to the simple message format and protocol exchange that leads to a reduced parsing and processing complexity, saving system resources as a consequence (CPU, memory, battery, *etc*.). On the other hand, the discovery of CoAP services and resources is also important since it is expected that the number of devices and services offered by these devices will grow rapidly [75]. Additionally, UDP binding in CoAP provides optional reliability when supporting unicast and multicast requests.

CoAP defines an *endpoint* as an entity that participates in a CoAP exchange. A CoAP message has a 4-byte header and, optionally, a set of options and a payload. Each message contains a *Message ID* used to detect duplicates and for optional reliability. A message can be of a type (Type field): *Confirmable* (CON), *Non-confirmable* (NON), an *Acknowledgment* (ACK) or a *Reset* (RST). CON or NON messages can be *requests* or *responses* depending on the *Code field* value in the header. An endpoint sending a Confirmable message will first wait for an ACK message. To improve the efficiency (by sending more information with fewer messages), a *piggybacked response* can be included in the payload of an ACK to answer a Confirmable request. In contrast, if a Non-confirmable message is sent, an ACK message is not expected.

A *CoAP client* is an endpoint that sends *requests* to a *CoAP server* for a service. When the CoAP server receives a request, it may send a response. A token value (*Token*), which is chosen randomly (although, it could be zero length) is used to relate a request to the corresponding response.

If the CoAP server cannot process a message, it will send a *Reset* (RST) message to indicate its inability to process it. This might happen, for example, when the smart object reboots and has lost some state to process the message. The RST can also be used to perform aliveness test of an endpoint, also called *CoAP Ping*. If the server is going to answer a Confirmable message with a CoAP response, but the information is not available yet, it can

send an ACK message with empty payload (a CoAP message containing only the header) to indicate that the response will arrive later.

CoAP defines four basic request methods *GET*, *POST*, *PUT* and *DELETE*. The client can use the GET method to retrieve information from the server. It can also use the POST method to create a new resource in the server. The server will then assign an identifier to the created resource. From that point, the client can use the POST or PUT method to update it. Finally, DELETE is used to erase a resource in the server.

2.4 Bootstrapping protocols

In this section we review a protocol related to bootstrapping in IoT, PANA and 802.1X. PANA is currently used for bootstrapping in IoT, and is the only one classified as bootstrapping protocol by the IEEE 802.15.9 standard, which is consistent with the definition of bootstrapping we use here, which follows the schema of the EAP KMF. In the other case, 802.1X is adapted to be used in IoT, but as we will see, it provides a link-layer dependent solution.

2.4.1 Protocol for Carrying Authentication for Network Access (PANA)

The Protocol for Carrying Authentication for Network Access (PANA), designed by the Internet Engineering Task Force (IETF), is an EAP lower layer designed to transport EAP messages on top of IP, using UDP as transport. PANA carries EAP messages to support different authentication mechanisms for network access. PANA is considered a bootstrapping protocol, as stated in the IEEE 802.15.9 [5], and is differentiated from other protocols such as IKEv2 or HIP-DEX, which are not considered bootstrapping protocols. This distinction is made in IEEE 802.15.9 due to the fact that PANA is used in that specification to provide link layer credentials (LLC), which is consistent with our broader view of bootstrapping, where we consider that it can be also used to generate key material for various security association protocols.

The architecture, illustrated in Figure 2.4, defines five entities: The Smart Object, in this case is the *PANA Client* (PaC). A *PANA relay* (PRE), that can be another smart object that is already joined the security domain. An *Enforcement Point* (EP) and a *PANA Agent* (PAA) can be located in non-constrained devices. The PaC requests access to the network service that is managed by an EP. The EP can be a router or an access point, controlled by the PAA. The PAA handles the authentication and authorization of the PaC for the network service. To accomplish this task, the PAA communicates with another entity, a AAA server

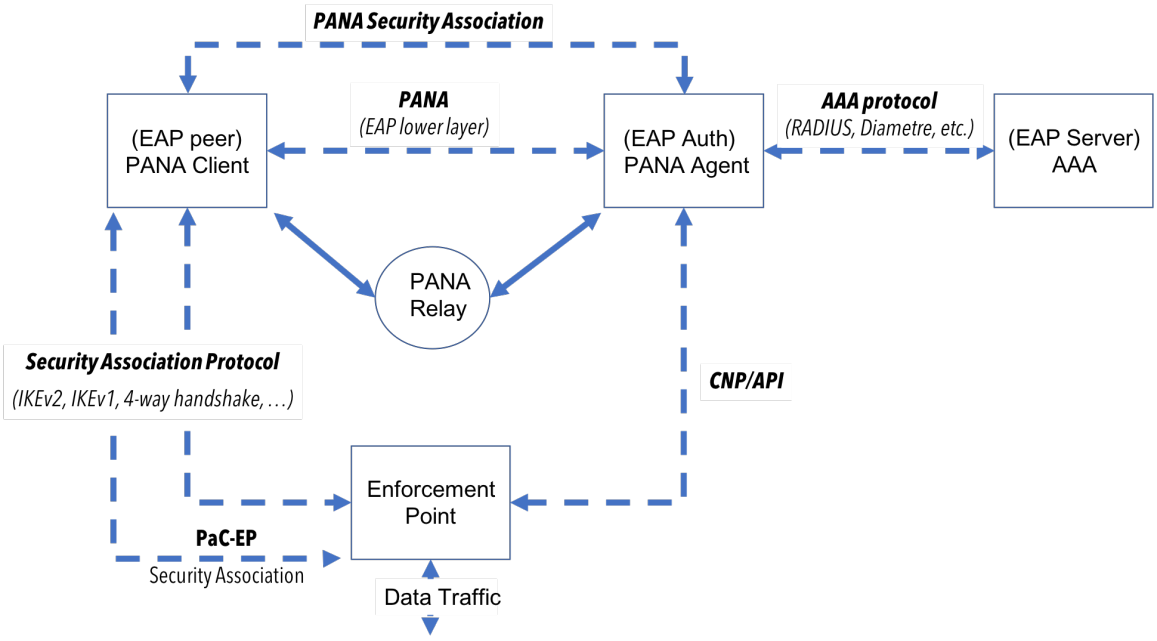


Fig. 2.4 PANA architecture

that authenticates the PaC, verifying the credentials of the PaC and sends the authorization information parameters (e.g., cryptographic material, network access lifetime, quality of service (QoS) filters, etc.) to the PAA. The the PAA sends the EP some configuration information through a *Configuration Network Protocol* (CNP) or *Application Program Interface* (API) like *Simple Network Management Protocol* (SNMP) [118].

According to this operation, depicted in Figure 2.4, the PaC, PAA and AAA implement the EAP peer, EAP authenticator and EAP server functionalities, respectively. The PRE entity intervenes when the PaC is not able to reach the PAA, relaying every message from the PaC to the PAA and to the PaC from the PAA.

The PANA general flow of operation, illustrated in Figure 2.5. The communication is done between the PaC and the PAA through the PRE and an EP. Each message sent from the PaC is sent to the PRE, which encapsulates this message into a new type of message called PANA Relay message (PRY), containing the Information of the PaC (IP and port) and the original message. This message arrives at the PAA, through the EP, which decapsulates it and continues the conversation with the PaC through the PRE, using PRY messages.

The exchange starts with the PaC initiating the exchange with a message to trigger the start of the authentication process (PCI). The PAA upon reception of this message sends a *PANA Request* (PAR) to request the identity of the PaC. This message contains an EAP Request/Identity Message. The PaC responds in a *PANA Answer* (PAN) with an EAP Response/Identity. With the identity, the PAA sends the message to the AAA Server, which

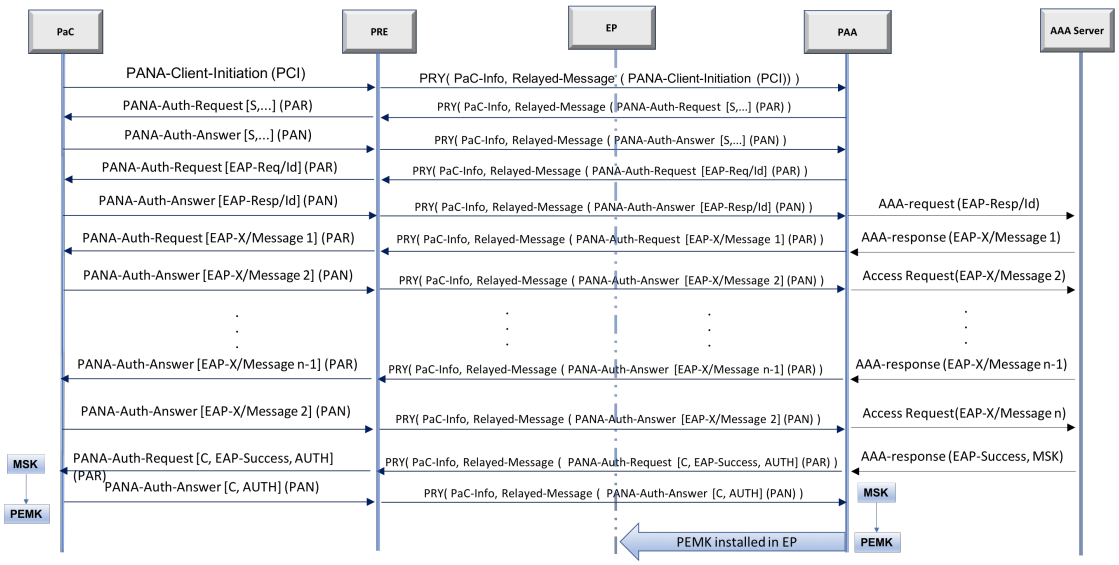


Fig. 2.5 PANA message flow in pass-through mode (with generic EAP method X)

decides the EAP method to be used, and starts said EAP method exchange with the PaC, whilst the PAA acts as a forwarder of these messages. When the EAP authentication is completed successfully, the AAA server and PaC derive the EAP key material (the MSK is sent to the PAA through the AAA protocol). The MSK is used to establish a PANA Security Association with key material derived from the MSK, and the PaC and PAA authenticate each other. Additional key material can be derived to secure the communications between other entities. In this case there is a key, called *PaC-EP Master Key* (PEMK), that is sent from the PAA to the EP, to protect the communications between the PaC and the EP.

2.4.2 IEEE 802.1X

IEEE 802.1X[1] is an IEEE standard for *Port-based Network Access Control* (PNAC). IEEE 802.1X defines the encapsulation of the Extensible Authentication Protocol (EAP) over IEEE 802, known as EAPoL. It provides an authentication mechanism to devices wishing to join a LAN or *Wireless Local Area Network* (WLAN). Protocols such as IEEE 802.11 [6], establish the security association using key material as a consequence of running the 4-way handshake. For the IEEE 802.1X model to work, it defines an EAP lower layer known as EAPoL to specify how to encapsulate the EAP packets into 802.11 frames and the message exchange between the EAP peer and the authenticator.

The architecture illustrated in Figure 2.6 defines 3 entities: The *supplicant* (EAP peer) that intends to join the network, the *authenticator* (EAP authenticator) that provides the

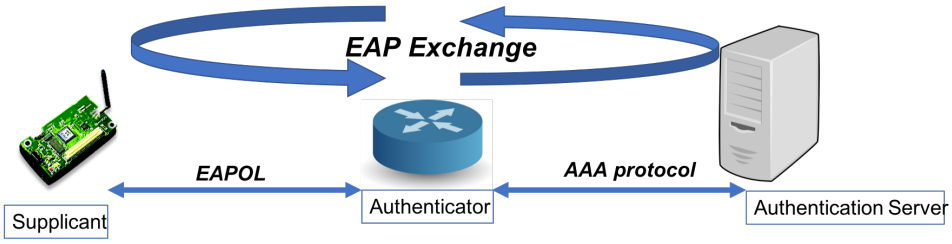


Fig. 2.6 EAPOL architecture

authentication service, and the *authentication server* (EAP server) that performs the authentication.

The general flow of operation of the protocol starts with the communication port of the authenticator in the unauthorized state. The supplicant initiates the communication by sending the *EAPOL Start* message to the authenticator. This means that the only traffic allowed is related to 802.1X. When the authentication is successfully completed, the port state changes to authorized, and the client is granted access to the network. One of the downsides is that 802.1X uses EAP just for authentication, the key material that is exported from the authentication process is not used to protect the communications. To accomplish this other protocols have to be used. In wired networks, the IEEE 802.1AE (also known as MACsec) [8] is used to encrypt the communications, and in wireless networks WPA 2-enterprise can be used for this purpose [62].

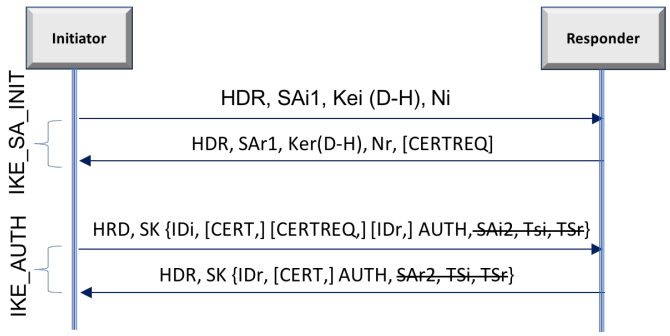
802.1X is proposed in several works in the context of IoT, such as in Hernández-Ramos *et al.* [85], Pawlowski *et al.* [150], Liu *et al.* [119]. The problem with link-layer dependent solutions is that, even though they are optimized, they are not applicable to other link-layers and it is not defined how the derived key material is used (if any).

2.5 Security Association protocols

In this section we review the Security Association Protocols (SAP) used in IoT. These protocols correspond to the EAP KMF phase 2, once the bootstrapping has finished and key material is derived and a unicast or multicast security association can be established between the Smart Object and the Controller. In this context, our bootstrapping service could be used to provide the key material needed to run these security association protocols.

2.5.1 Internet Key Exchange Protocol version 2 (IKEv2)

The Internet Key Exchange Protocol version 2 (IKEv2)[105] is a component of IPsec and is used to perform mutual authentication and establish and maintain security associations (SAs).



Notation	Meaning
AUTH	Authentication data
CERT	Certificate
CERTREQ	Certificate Request
HDR	IKEv2 header
IDi	Initiator Identifier
IDr	Responder Identifier
KE	Key Exchange (Diffie Hellman parameters)
Ni, Nr	Nonces
SA	Security Association
SK{X}	X is Encrypted and Authenticated
TS	Traffic Selector

Fig. 2.7 Minimal IKEv2 Exchange for IEEE 802.15.9

It creates and maintains a state that defines, among other things, the cryptographic algorithms that will be used, the specific services that are provided to the datagram, and the keys used as input of the cryptographic algorithms. IPsec services such as confidentiality, data integrity, access control and data source authentication for IP datagrams are possible thanks to the maintenance of a shared state between the source and sink of IP datagrams, which is done with IKEv2. It runs on top of UDP (port 500) and allows authentication with symmetric key (PSK), certificates or raw public keys, and EAP. IKEv2 follows a client-server architecture: The entity that starts the exchange is called the *initiator* (client); the entity that answers the initiator is the *responder* (server).

IKEv2 as protocol for establishing and maintaining security associations is considered in IoT, such as securing IEEE 802.15.4 [160], and it is one of the KMP depicted in IEEE 802.15.9 [5] to secure IEEE 802.15.4. To be able to use IKEv2 in constrained devices, the RFC7815 [111] defines a minimal version of the IKEv2 protocol. This minimal implementation only supports the initiator end of the protocol. Also, minimal IKEv2 only supports the initial IKE_SA_INIT and IKE_AUTH exchanges. It does not initiate any other exchanges and replies with an empty (or error) message to all incoming requests. To reduce the overhead in the protocol implementation, most of the optional features of IKEv2 are not implemented such as NAT traversal, *Internet key exchange* (IKE) SA re-key, Child SA re-key, multiple Child SAs, deleting Child / IKE SAs, Configuration payloads, Extensible Authentication Protocol (EAP) authentication, COOKIEs, etc.

This implementation of IKEv2 only uses the first two exchanges, called IKE_SA_INIT and IKE_AUTH with the objective of creating the first Child SA to protect IKE messages. In the first exchange (IKE_SA_INIT), the initiator starts the negotiation of security parameters for the IKE SA exchange the nonces and its part of the Diffie-Hellman exchange. The second exchange (IKE_AUTH), is used to authenticate both parties, to exchange identities

and certificates and establish the first security association (SA) that is called in IKE a child security association (CHILD_SA). IKEv2 can also use EAP for authentication. The exchange of EAP messages is done using several IKE_AUTH exchanges, with the initiator being the EAP peer and the responder the EAP authenticator. However, the keys derived from the EAP authentication process are used to generate the IPsec SA keys.

In IEEE 802.15.9, IKEv2 is used to establish a secure link establishment referred to as a link-establishment KMP. We note here that, in 802.15.9, IKEv2 is not described as a bootstrapping KMP. IEEE 802.15.9 specifies that depending on the environment where IKEv2 is used, only some of the authentication methods supported by IKEv2 are needed and that, in order to indicate which optional features of IKEv2 are used in a particular environment, it can be specified in a profile for that environment. For the specific case of IEEE 802.15.4, IEEE 802.15.9 specifies that, of the features of IKEv2, they do support the basic IKEv2 exchange, IKE_SA_INIT, and IKE_AUTH [106], the Childless Initiation of IKEv2 [141]. They also use Authenticated Encryption Algorithm with the Encrypted payload with IKEv2 [29]. They do not support the negotiation of multiple protocols within the same proposal, the capability to handle multiple outstanding requests, Cookies, Configuration Payload or NAT-Traversal. Furthermore, they do not use UDP, the negotiation is transported in IEEE 802.15.4 Information Elements (IE). Figure 2.7 illustrates the minimal IKEv2 exchange in 802.15.9.

In the IEEE 802.15.9 standard it is also specified an alternative use of IKEv2 to their minimalistic version if IKEv2, to support enterprise or large-scale IKEv2 use cases. It is considered the use of AAA infrastructures, including performing an EAP authentication, since it is supported by IKEv2. This process is intended when the devices are going to establish IPSec, which is recognized too heavyweight to constrained devices. In fact, in Marin-Lopez *et. al* [157], IKEv2 is compared with PANA for exactly this purpose, because both are defined on top of UDP and support EAP authentication to work with AAA. They conclude that IKEv2 is not a replacement for PANA for two reasons. First, because IKEv2 mandates a Diffie-Hellman key exchange, considered more expensive than other cryptographic operations (e.g., Hash-based message authentication code (HMAC)), specified in RFC 5191 for the PANA SA. Second, an IPsec SA is always created in [106], which is not required in many access networks. For these reasons, IKEv2 is not considered as a replacement for PANA for network access authentication.

This is why throughout this dissertation, we will compare with PANA only, as it better than IKEv2 for network access authentication.

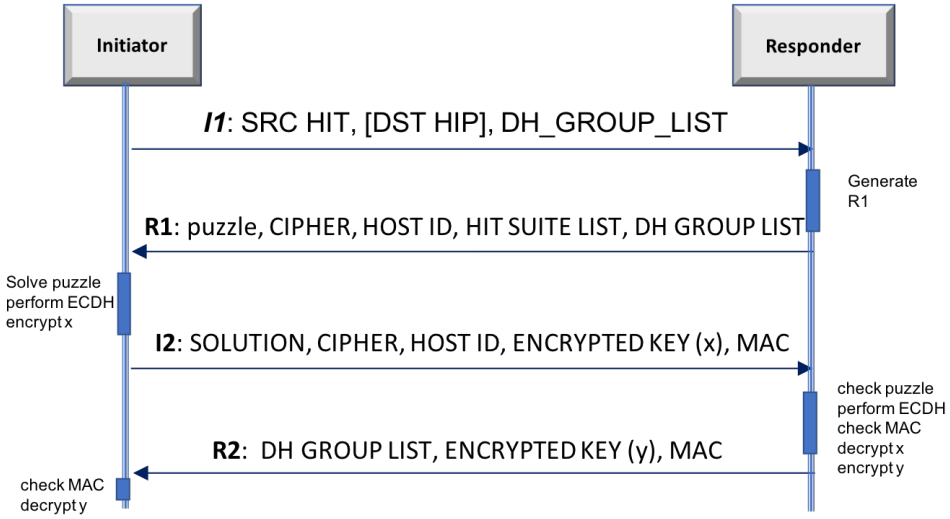


Fig. 2.8 HIP-DEX Exchange

2.5.2 Host Identity Protocol Diet Exchange (HIP-DEX)

The Host Identity Protocol Diet Exchange (HIP-DEX) [132] is designed as an end-to-end authentication and key establishment protocol, specifically designed for computation or memory-constrained devices.

HIP-DEX follows a client-server architecture, called initiation and responder respectively. The exchange consists of 4 messages. These four messages are *I1, R1, I2* and *R2*. Figure 2.8 illustrates the HIP-DEX protocol message flow.

The first message, *I1*, includes the source host identity tag (*HIT*) (a 128-bit value encoding a hashed encoding of the Host Identifier) and the destination *Host Identity Tag* (HIT) optionally if it is known. I1 also initializes the negotiation of the Diffie-Hellman (DH) group used for generating the Master Key SA. The second message, *R1*, contains a puzzle (i.e., a cryptographic challenge) intended for the initiator. R1 also specifies the Diffie-Hellman parameter and the supported cryptosuite of the Responder. The third message, *I2*, sends the solution of the puzzle, plus a key wrap parameter (cyphered with the session key from the DH exchange) that carries secret keying material of the Initiator that will constitute half of the final session key. This message is integrity protected with a Message Authentication Code (MAC). The last message, *R2*, acknowledges the receipt of the I2 packet. R2 contains another key wrap parameter containing the other half of the final session key and it is also MACed. The session key is generated by the key wrapped parameters found in messages I2 and R2.

HIP-DEX is a lightweight alternative that is considered for IoT such in [90] and it is one of the KMP depicted in IEEE 802.15.9 [5] to secure IEEE 802.15.4. Furthermore, especially

designed for devices with very high constraints, there is a version of the HIP-DEX protocol that uses pre-shared keys HIP-DEX PSK, proposed by Garcia-Morchon *et. al* [68].

2.5.3 Datagram Transport Layer Security (DTLS)

The DTLS [162] arises from the need to provide security to the communications to new application layer protocols that work on top of UDP, and cannot use TLS as it works on top of TCP. DTLS is designed to be as similar as possible to TLS, with the necessary adaptations to work on top of UDP. DTLS allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery providing equivalent security guarantees as TLS. DTLS can be divided in two protocols the *Handshake Protocol* and the *Record Layer*. The handshake protocol is responsible for allowing peers to agree upon security parameters for the record layer, authenticate themselves, instantiate negotiated security parameters, and report error conditions to each other. The Record Layer uses the security parameters generated by the Handshake Protocol to secure the communications. To authenticate the peers, DTLS can use *Raw Public Key* (RPK), Certificates or PSK.

DTLS has two main differences with respect to TLS. The first is, DTLS has to take care of reliability, which TLS already provides by working on top of TCP. The second is, that there is an inter-record dependency, for which stream ciphers cannot be used. To solve these issues, the changes that are implemented are: For dealing with packet loss, DTLS adds a retransmission timer, for reordering, a specific sequence number is added to the record protocol. Reply detection is done with a bitmap window of the received records. To support fragmentation, DTLS adds fragment offset and fragment length.

The architecture of DTLS defines two entities: a client and a server. The message flow starts when the client sends the *client hello* message. This message is then processed by the server and responds with a message containing a cookie, used to prevent denial of service attacks. Then the client sends, again, the client hello message, but this time with the cookie received from the server. After this initial exchange, the server and client exchange their certificates and negotiate the cipher suite, after which the handshake is finished and a security association between the client and server is used to secure their communications through the record layer.

DTLS is the current standard to protect CoAP exchanges [190]. Due to the heavyweight process of handshake needed in DTLS and the fact that DTLS breaks when the communication is used between CoAP endpoints through proxies or intermediaries, there are alternatives that are being developed to cope with these drawbacks. An example of this is the OSCORE protocol, which we will review in Section 2.5.5.

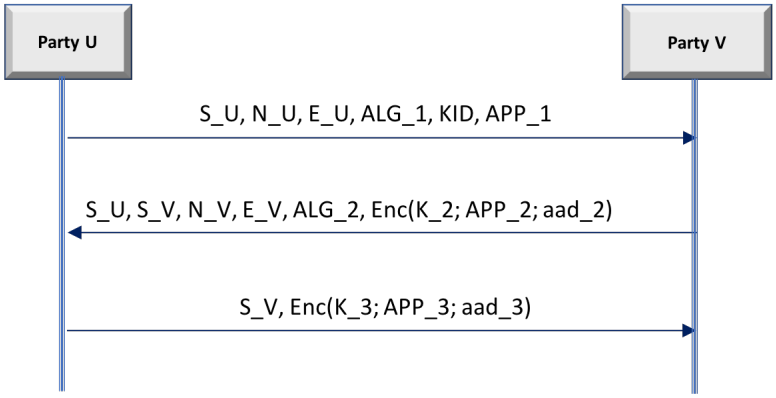


Fig. 2.9 EDHOC message flow with symmetric key authentication

Additionally, in DTLS there is a proposal to support multi-hop topologies by Kumar *et. al* where they define a new DTLS entity, the DTLS relay [108]. They define two variants of this entity: a stateful and stateless mode. The stateful mode stores information about the smart object that engages in the DTLS handshake, so avoiding sending this information over the network. In the stateless mode, the DTLS relay does not store any information related to the exchange. The IP information of the smart object is sent along with the DTLS handshake message in a new message defined for the this relay entity: the DTLS relay message (DRY).

2.5.4 Ephemeral Diffie-Hellman Over COSE (EDHOC)

The *Ephemeral Diffie-Hellman Over CBOR Object Signing and Encryption (COSE)* (EDHOC) protocol is used for authentication and key establishment in IoT. EDHOC describes an authenticated Diffie-Hellman Exchange with ephemeral keys to be used over any layer [184]. The EDHOC messages are encoded using the *Concise Binary Object Representation* (CBOR) and CBOR COSE. The authentication can be done using credentials established out of band, e.g. from a trusted third party, using PSK, RPK, and X.509 certificates (Cert).

The architecture defines 2 entities in a client-server exchange: The client is called *U* and the Server is called *V*. EDHOC uses Ephemeral Diffie-Hellman (EDH) to generate a shared secret between both parties. This shared secret is different every time the protocol runs because the Diffie-Hellman (DH) parameters are generated every time. Figure 2.9, illustrates the case with authentication with symmetric keys. The message exchange is composed of 3 message:

The first message contains a session identifier (S_U), a nonce (N_U), the ephemeral public key of U (E_U), the supported cryptosuites (ALG_1). KID is used to identify which Pre-Shared Key to use. The second message contains the session identifier of party U (S_U) and a new session identifier of V (S_V), another nonce (N_V), the ephemeral public key of

V (E_V), and the chosen cryptosuite (ALG_2). It also contains a COSE structure ciphered with the key material that was agreed. The third message contains the session identifier of V (S_V) and a COSE object that is encrypted with the previously generated key material.

This protocol is intended to be used in constrained networks and nodes for authentication and key agreement. It is proposed in 6TiSCH [164] as an alternative to DTLS, to create a security association and protect the communications with OSCORE. The issue is that, like DTLS, we need to have key material to run the security association protocol. This issue is not addressed in the 6TiSCH minimal security framework [206], as they consider it is outside of the scope of their document.

2.5.5 Object Security for Constrained RESTful Environments (OSCORE)

The OSCORE is a security protocol is created to secure the communications between two CoAP endpoints. This protocol provides protection at application level, achieving end-to-end protection. The reason for this is that the standard protocol to secure communications with CoAP is DTLS and CoAP endpoints may communicate through proxies or intermediaries that would break the DTLS channel used to protect CoAP exchanges. To achieve this, OSCORE defines a new CoAP option called Object-Security Option that is used to protect the information that is not intended to be seen by the intermediaries or proxies. OSCORE builds upon COSE [181], providing end-to-end encryption, integrity, replay protection, and binding of response to request.

OSCORE requires the establishment of a a shared security context between the client and server that is used to process the COSE objects. OSCORE uses COSE with *Authenticated Encryption with Associated Data* (AEAD) [125], algorithm for protecting the messages. The Security Context in OSCORE is composed of a "Common Context", a "Sender Context", and a "Recipient Context". The CoAP endpoints protect messages to send using the Sender Context and verify the messages they received using the Recipient Context. Both contexts are derived from the Common Context and additional data. The Common Context contains the following input parameters: The algorithm used for encryption (AEAD Algorithm), a *Key Derivation Function* (KDF), Master Secret and Master Salt to derive further key material, and a common Initialization Vector (IV) for the encryption algorithm. The input parameters that generate the security context are pre-established, and how they are established is application specific. To generate context we can use our bootstrapping service to protect the communications between the Smart Object and the Controller.

2.5.6 Minimal Security Framework for 6TiSCH

The Minimal Security Framework document [206] describes the minimal framework required for a new device that is called "pledge", to securely join a 6TiSCH network(IPv6 over the TSCH mode of IEEE 802.15.4e). This framework follows a 3-entity architecture: a pledge (the smart object), a Joining Proxy (an intermediary that implements a stateless CoAP proxy) and a Join registrar / coordinator (the Controller).

They assume a one-touch scenario, where the smart object is provisioned with a Pre-Shared Key (PSK) and, optionally, a network identifier, before attempting to join the network. The same parameters are provisioned to the Controller. With a single CoAP exchange (request-response) protected by OSCORE, the pledge requests admission into the network and the *Join Registrar/Coordinator* (JRC), after evaluating the requests positively, sends the link-layer keying material and a short link-layer address in the response.

In the exchange to join the network between the pledge and JRC, illustrated in Figure 2.10, 6JP stands for 6TiSCH Join Protocol. Next we elaborate the details of the exchange.

1. The pledge is listening for an Enhanced Beacon (EB) frame (IEEE 802.15.4-2015). This EB provides information about network synchronization, which tells the pledge when it can send a frame to the *Joining Proxy* (JP) and when it can expect to receive a frame.

2. The pledge configures its link-local IPv6 address and announces it to the JP.

3. The pledge sends a Join Request to the JP, so identifying itself securely to the network. The Join Request is forwarded to the JRC. The JRC can be co-located on the JP or another device.

4. If the request is processed successfully, the pledge receives a join response from JRC (through the JP), which sets up one or more link-layer keys that are used to authenticate and encrypt subsequent transmissions to peers, and a short link-layer address for the pledge.

They leave out of scope how these key materials and the OSCOAP security context are generated. This provisioning process can be carried out by our bootstrapping service, which will enable the joining of a pledge into a 6TiSCH network.

2.5.7 Enrollment over Secure Transport (EST) over CoAP

Enrollment over Secure Transport (EST) [153] is used as a certificate management protocol that works on top of HTTPS. EST is used for authenticated/authorized endpoint certificate

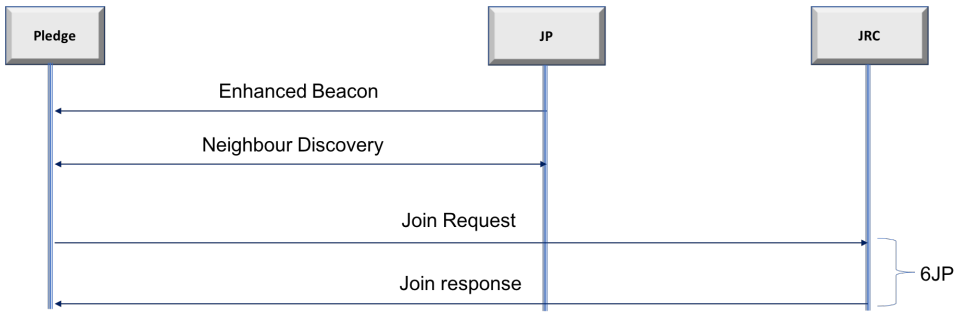


Fig. 2.10 Overview of a 6TiSCH join process.

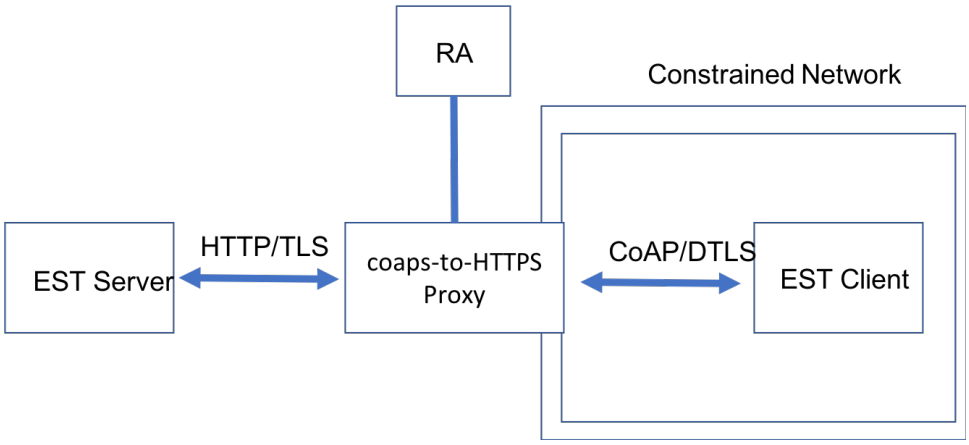


Fig. 2.11 EST over CoAP architecture

enrollment (and, optionally, key provisioning) through a Certificate Authority (CA) or Registration Authority (RA). This functionality is also interesting for constrained environments and devices for which the adaptation of EST over CoAP has been designed [47]. Examples of the use cases of EST in constrained networks are secure bootstrapping and certificate enrollment.

The adaptation of EST to work in constrained network and devices entails the mapping of some of the protocols used in the original version to their counterparts for constrained networks. HTTP is mapped to CoAP to transport the EST messages. The secure transport, in the original EST standard, is mapped to DTLS, the standard protocol to protect CoAP exchanges. As can be seen in Figure 2.11, to maintain interoperability, there is a proxy between both the constrained and non-constrained networks, that translates CoAP to HTTP at the same time that it is the endpoint of both DTLS (for the constrained network) and TLS (for the normal network).

Although this adaptation of EST for constrained networks assumes its feasibility in these scenarios, we have to consider two drawbacks of this approach. The first is that it uses DTLS, which is already considered heavyweight for some IoT (e.g., LPWAN) networks due to its

handshake. Following the use of DTLS, and as the authors themselves recognize, the proxy must be deployed with great care, because secure connections are broken in the proxy. The second issue with this option is that they limit the bootstrapping to devices that have the capabilities to handle public key cryptography, because the use certificates with EST is the only option available.

This approach is the one used in BRSKI [152], and the authors recognize that their solution is aimed at non-constrained devices and networks.

2.6 Work in standardization Groups related to bootstrapping in IoT

Among the different aspects of security, the importance of bootstrapping in IoT has been highlighted in several works on the IETF [209, 79]. The problem of bootstrapping has been discussed in several IETF *Working Group* (WG) such as 6TiSCH, ACE WG, CoRE WG and *IPv6 over Networks of Resource-constrained Nodes* (6lo) WG. Other standardization organisms and alliances such as IEEE, *World Wide Web Consortium* (W3C), OMA, *European Telecommunications Standards Institute* (ETSI) and IPSO, are working in the IoT and some of them provide insights into the bootstrapping process in the IoT.

We review next some of the work done in this regard by the aforementioned standardization organizations.

2.6.1 Internet Engineering Task Force (IETF)

The IETF has several working groups involved in the development of protocols related to the Internet of Things. The 6LoWPAN WG defines an adaptation to use IPv6 in IEEE 802.15.4, the CoRE WG provides protocols such as CoAP, see section 2.3.3, and OSCORE (see section 2.5.5) to provide e nd-to-end security for constrained devices. The 6TiSCH WG [95] takes a cross layer view in their architecture, considering aspects form the scheduling of the communications, routing and bootstrapping of a 6TiSCH network. The ACE WG takes a more specific , focusing on the field of security for constrained nodes and environments. Their aim is to produce a standardized solution for authentication and authorization to "enable authorized access to resources identified by a *Uniform Resource Identifier* (URI) and hosted on a resource server in constrained environments." Currently they propose a framework using OAuth to manage authentication and authorization, using CBOR *CBOR Web Token* (CWT) to transport claims, that is, what the bearer or the token is authorized to do. Another contribution is the development of what they are called profiles; the necessary adaptations to

include different protocols into the ACE framework, which include OSCORE and DTLS, among others. Proposals for novel security association protocols for IoT can be found in Ephemeral Diffie-Hellman Over COSE (EDHOC) [184], a protocol to run an authenticated Diffie-Hellman key exchange. The ANIMA WG, defines a common infrastructure for certain functions like discovery, node identification, negotiation, transport and security mechanisms in autonomic networks. Among their works, there are documents treating the bootstrapping problem [152]. Although, they target non-constrained networks and devices as stated in their own documents and we aim to design a bootstrapping service that can be used in constrained devices and networks.

2.6.2 Institute of Electrical and Electronics Engineers (IEEE)

The process of bootstrapping in IEEE 802.15.4 is eased by using IEEE 802.15.9, where it is defined how to transport KMP datagrams over IEEE 802.15.4. Guidelines are provided on how to proceed with some current standards such as PANA, Dragonfly, 802.1X, etc. Curiously enough, in the IEEE 802.15.9 document there is a clear distinction between PANA and the other KMP, stating that PANA is used for bootstrapping, which they define as the process of pre-provisioning link-layer credentials.

While this initiative optimizes the bootstrapping process by transporting the protocols at link layer, looking at the big picture, there is a variety of radio technologies and MAC layers, and recently there are new ones considered in LPWAN. This translates into every technology performing a similar effort when pursuing an optimized bootstrapping protocol. This is why in this thesis we provide a generic solution to bootstrapping to the IoT. The results of a generic solution are not expected to improve these kinds of solutions that are optimized at link layer. Instead, we offer a trade-off, with a generic solution that would aid in managing the bootstrapping process.

2.6.3 3rd Generation Partnership Project (3GPP)

The 3GPP is involved in the design of what is known as 5G (Fifth Generation of mobile communications standard). It is commonplace to see 5G coupled with IoT and it is not surprising, since the advent of 5G communications represents a potentially disruptive element to IoT. With an increased data rate, a reduced end-to-end latency, and an improved coverage with respect to 4G, 5G holds the potential to enable even the most demanding IoT applications. Additionally, 5G focuses on the integration of heterogeneous access technologies, and may play the role of a unified interconnection framework, facilitating a seamless connectivity of "things" to the Internet.

To support the large number of devices that are expected to be present in IoT, there are different proposals and evaluations of the different technologies to carry out this task. Current work in the 3GPP is considering EAP for authentication [1] [137]. This goes along the same line of this thesis, in the sense that we understand that integration of large scale deployments under domains that may not be managed by the owner of the devices, most probably a *Telephone Company* (TELCO). This brings to the table the need to provide flexibility when choosing an authentication method and the need to provide Identity Federation support for bootstrapping those devices. The use of AAA is being considered in technologies such as LPWAN with support for large number of devices, and it is discussed in the context of 5G with supporters [56] and detractors [200], weighing the pros and cons of the use of AAA in 5G. We can expect that, being the cons mostly related to technical issues, and vulnerabilities of current implementations, we see that the underlying concept of AAA is still valid and can be considered to be used in this context.

2.6.4 Other Standardization Groups

There are other standardization groups, such as the W3C, the IPSO, OMA etc. that are involved in the development of new standards for the Internet and the IoT. We review some of their work related to security below.

The *Zigbee IP* [217] standard uses PANA for the bootstrapping protocol. In particular, the standard proposes the use of a PANA and EAP-TLS method [193] (based on X.509 certificates) for the EAP authentication. They use EAP in standalone mode without AAA infrastructures. Nevertheless, our CoAP-based EAP lower-layer can be considered instead of PANA, as we will see throughout this dissertation. The OMA [158] defines a protocol for managing IoT infrastructures called (OMA *Lightweight M2M* (LWM2M)). Among other aspects, bootstrapping is defined using CoAP and DTLS. The architecture consists of three entities, *LWM2M Client*, *LWM2M Server* and *Bootstrap Server*. It is specified that the Client and the Server share credentials with the Bootstrap Server in order to authenticate, but it is not specified how the credentials are configured. The *IPSO Alliance* [189] defines the use of CoAP as application protocol, using DTLS to protect the devices with sensible resources such as actuators unless other underlying security mechanism are used. No further considerations are added to the security and bootstrapping landscape. Finally, the W3C [4] considers the case of security on the smart object bootstrapping outside its scope, proposing the adoption of other security implementations such as the *ARM Trust Zone* [18], although, there is a proposal also aimed at using *Generic Bootstrapping Architecture* (GBA) [86].

[1] http://www.3gpp.org/ftp/TSG_SA/WG3_Security/TSGS3_89_Reno/Docs/S3-173155.zip

2.7 Conclusions

This chapter has surveyed the protocols and technologies most relevant to this dissertation, detailing their main purpose, how they work and their relation to bootstrapping in the IoT. Moreover, this chapter introduces the technologies that are used as bases for this dissertation, namely AAA infrastructures, the Extensible Authentication Protocol (EAP) and the CoAP. We have also explained the protocols that are currently used for bootstrapping in IoT, or related to it, because they will be referred to throughout this dissertation as being categorized as part of the SAPs within the *EAP Key Management Framework* (EAP-KMF). Although, they are completely valid for the purpose for which they were designed, they do not address all the objectives and requisites covered in this thesis and described in Chapter 1. Hence, this document defines a new bootstrapping service to provide a better solution to the stated problem. The following chapters of this thesis will provide the details of each contribution and will analyze future work.

Chapter 3

Lightweight CoAP-Based bootstrapping service for the IoT: CoAP-EAP

This chapter introduces the first of the main contributions in this thesis, for the bootstrapping service for LR-WPAN named *CoAP-EAP*. Here, we summarize the motivation for designing a bootstrapping service for large-scale IoT deployments and review, now in more detail, the related work in the area of bootstrapping in IoT. After that, we describe the bootstrapping service, its design, the architecture and general flow of operation. We also specify how key material can be generated to perform different security association protocols betwen the *Smart Object* and the *Controller* using, as an example, DTLS. Finally, we show the experimental results and how CoAP-EAP compares toPANA, one of the current bootstrapping protocols in IoT.

3.1 Introduction

Over the last few years, the global information network formed by Internet-connected objects, known as the IoT [75], has undergone impressive growth. To accomplish the vision of the Internet of Things, standardization organizations and the research community have been working on the definition of several architectures and protocols [98, 147]. An important part of the IoT networks is foreseen to be formed by a huge amount of devices with constrained capabilities (called smart objects) and IP-based networking connectivity [202].

These are typically based on low power radio technologies [202] such as IEEE 802.15.4 [92] or Bluetooth Smart [71]. For IP-based communications, the IP version 6, over 6LoWPAN standard [82] enables IPv6 connectivity for smart objects. This brings in new and promising areas of application, such as smart cities, smart grids, home automation, e-healthcare, among

others. While it allows for opportunities and improvements in our daily life, it also raises multiple security risks, which need to be tackled. In particular, one of the basic aspects of security is the bootstrapping of the smart object [36]. As described by Garcia-Morchon *et al.* [70] bootstrapping refers to the process by which a smart object securely joins the IoT network at a given time and location. It includes the authentication and authorization of a device as well as the transfer of security parameters (e.g., keying material) to the device to allow a trustworthy operation in a particular network. As a consequence of the process, the node joins a security domain and accounting of the bootstrapping service usage may be carried out.

In very demanding IoT scenarios, such as smart cities, a high number of smart objects will need to be authenticated and authorized before joining a security domain. The bootstrapping service is in charge of these operations. Additionally, these smart objects may belong to different organizations yet be deployed and bootstrapped in the same security domain. Thus, the concept of identity federation becomes relevant, though it remains unclear how it will be managed [165].

As described in Chapters 1 and 2, the current state of the art in the area of bootstrapping in IoT does not account for a set of requirements that allow for a bootstrapping solution for large-scale IoT deployments. Some of the solutions are optimized for specific link-layer technologies and are not usable in other link-layer technologies. Other solutions use protocols that are too heavyweight for highly constrained devices and networks. Some do not account for large-scale deployments, assuming pre-installed key material and running a security association protocol between the smart object and the Controller, so complicating the management of large scale deployments. Furthermore, identity federation is not considered in most cases, which limits the applicability to scenarios that do not consider that smart objects from different organizations might be deployed in the same security domain.

For these reasons, we foresee the key importance of AAA-based infrastructures [44] to provide a flexible, scalable and federation-aware bootstrapping service in the IoT. The reasons are two-fold: first, they are robust infrastructures for managing the authentication, authorization and accounting for the activity the smart objects and, in conjunction with the Extensible Authentication Protocol (EAP) [10], they provide a secure framework for flexible authentication, authorization and key distribution [88, 12]. Some evidence of the use of the AAA framework in the context of the IoT can be found in [81, 43, 2]. Second, they are widely used to manage a great number of device connections and, therefore, AAA support large scale deployments. In fact, AAA infrastructures based on the protocol Diameter [57] are commonly used in 3G networks to control the access of millions of users [172]. Another example is *eduroam* (educational roaming network) [212], which is a world-wide federation

for WiFi connectivity across campuses and research and educational organizations around the world that supports thousands of users. Eduroam deploys EAP and an AAA infrastructure based on the *Remote Authentication Dial In User Service* (RADIUS) [167], which provides the identity federation substrate.

In this context, we propose a novel bootstrapping service that is built on top of the *Constrained Application Protocol* (CoAP) [190] with the assistance of EAP and AAA infrastructures. The main reason for using CoAP is that it has recently been standardized as the application protocol for exchanging information between smart objects and, therefore, is specifically designed for devices with small memory and computational resources, such as those expected in IoT networks. In fact, our proposal stems from the (realistic) assumption that the smart objects will generally ship a CoAP implementation. Our service also assumes the presence of a centralized entity, the *Controller* (e.g., the coordinator in ZigBee IP [217]) which manages the access to a particular security domain and interacts with a smart object to perform the bootstrapping. That is, the controller authenticates and authorizes the smart object to become a member of the security domain.

To achieve this, both entities use CoAP to transport EAP packets for the authentication and to carry authorization. The controller can interface with a backend AAA infrastructure to complete the EAP authentication, perform the authorization steps related with the bootstrapping service and, optionally, account for the activity of the smart object in the security domain.

In this chapter, we present our bootstrapping service, named *CoAP-EAP*, its architecture, the design and the performance evaluation with implementation in the Contiki OS Cooja simulator [145], as well as comparison with PANATIKI [175], which represents the best case of bootstrapping solutions also using EAP and AAA. The rest of this Chapter is divided as follows. Section 3.2 gives the state of the art. Section 3.3 details the proposed bootstrapping service architecture and operation. In Section 3.4, we show performance evaluation including message size, bootstrapping time, memory footprint, the probability of finishing a bootstrapping (success percentage) and energy consumption. Finally, we provide some conclusions and future work lines in Section 3.5.

3.2 Related work on bootstrapping in IoT

In IoT landscape, where the configuration process of the smart objects is expected to be as much automated as possible, the benefits of a bootstrapping service are important: it can provide the necessary information to the smart object when it is deployed in an easy and automated manner making easier the scalability of the deployments.

T. Heer *et al.* [81] and Garcia-Morchon *et al.* [70] describe the concept of bootstrapping in IoT as the process of a smart object securely joining an IoT network at a specific place and time. It includes the *authentication* and *authorization* of the smart object as well as the transfer of some security parameters (e.g., keying material), so allowing a trustworthy operation in a particular domain. As a consequence o f the process, the smart object joins a security domain and *accounting* of the usage of the bootstrapping service may be carried out. The bootstrapping process is specially important in the case of IoT where the configuration of the smart objects is expected to be as much automated as possible, making the scalability of the deployments easier.

In general, the importance of bootstrapping in IoT has been highlighted in several works [209, 79]. On the one hand, the problem of bootstrapping has been discussed in several IETF WG such as 6TiSCH, ACE WG, *Constrained RESTful Environments* (CoRE) WG and *IPv6 over Networks of Resource-constrained Nodes* (6lo) WG. Additionally, an IETF mailing list has been created specifically to discuss the bootstrapping [97]. Other standardization organisms and alliances as IEEE, W3C, OMA, ETSI and IPSO among others, are working in IoT and some of them provide insights into the bootstrapping process in IoT.

On the other hand, there are several proposals discussing the general problem of bootstrapping for constrained devices while others also propose solutions that consider the use of EAP and, in some cases, the interaction with AAA infrastructures.

Garcia-Morchon *et al.* [70] analyze of the IP-based security protocols for bootstrapping in IoT networks. In the case of a centralized architecture, as our bootstrapping service, they highlight the potential use of EAP as a protocol to perform authentication and generation of fresh keying material. PANA [61] is proposed as a candidate to transport EAP between the smart object, acting as the *PANA client* (PaC), and the controller, which is the *PANA agent* (PAA) in PANA terminology. Nevertheless, these authors also recognize that the transfer of configuration parameters in a centralized scenario can be made by other protocols. In our solution, CoAP is used instead of PANA as EAP lower-layer.

O'Flynn *et al.* [144] discuss the general problem of bootstrapping for low-power wireless networks. They also consider a centralized architecture with a root trusted entity. They consider the option of EAP as authentication protocol and analyze PANA and IEEE 802.1X [1] as possible EAP lower-layers. However, as analyzed in [157], link-layer solutions, such as IEEE 802.1X, are unsuitable for multi-hop wireless networks. Additionally, He *et al.* [80] includes the possibility of using HIP-DEX [132] as a bootstrapping protocol, although this option does not have any interaction with EAP or AAA's, limiting the case to small or medium scenarios. Nevertheless, no concrete alternative is chosen in these works.

Sarikaya [180] and Sarikaya *et al.* [178] propose the use of EAP-TLS, based on certificates, as a specific method for authentication during the bootstrapping. The authors also consider PANA or IEEE 802.1X as EAP lower-layers.

S. Das *et al.* [43] propose a centralized alternative usingPANA, EAP and AAA to bootstrap a pre-shared key (PSK) to establish a DTLS [162] or IKEv2 [106] unicast security association between the smart object and the PAA (the controller). However, CoAP is used afterwards for the post-bootstrapping phase. In particular, the CoAP client (the smart object) requests the CoAP server to bring a key (*pull model*) from an *Authentication Server* (AS) which acts as EAP server (*i.e.*, AAA server) in the EAP authentication involved in the bootstrapping. Our solution does not require a specific protocol just for bootstrapping (PANA) but reuses the deployment of CoAP to build the bootstrapping service.

Moreno *et al.* [175] designed and implemented a lightweight version of a PANA client (PaC) for Contiki OS [52] (PANATIKI) by adapting PANA for constrained devices. It implies removing part of the PaC state machine to make it suitable for constrained devices. Although PANATIKI does not make any modification to the standard, it does not implement some parts of the standard PaC state machine. In other words, it represents a reduced version of the standard, and a best case for PANA-based solutions. That is why we will make a comparison with PANATIKI to evaluate our proposal against an implementation that is optimized for constrained devices.

It is important to note that (the use of) PANA is part of *Zigbee IP* [217]. In particular, the standard proposes the use of a PANA and EAP-TLS method [193] (based on X.509 certificates) for the EAP authentication. They use EAP in standalone mode without AAA infrastructures. Nevertheless, our CoAP-based EAP lower-layer could be used instead ofPANA.

Additionally, the *Institute of Electrical and Electronics Engineers* (IEEE) association in the standard IEEE 802.15.9 [5] proposes a transport method for KMP datagrams that will make use of existing KMP s with the IEEE 802.15.4 and.7. Guidelines will be provided regarding the use of KMP s like HIP, IKEv2, IEEE 802.1X and PANA. On another note, the *European Telecommunications Standards Institute* ETSI [55] defines the support for Generic Bootstrapping Architecture (GBA) and adopts PANA as an option.

It is worth mentioning there is also another set of solutions [68, 24, 112, 158, 189, 4] that show the need for a bootstrapping process but they do not consider EAP or AAA infrastructures as part of their solution. In this sense, they do not support federated authentication and authorization, so limiting the deployment to small or medium scale scenarios. For example, Garcia-Morchon *et al.* [68] propose two different architectures providing secure network access, key management and secure communications. The first solution uses a variant of

HIP-DEX based on pre-shared keys and the second solution uses DTLS. For secure network access, a pre-shared key is assumed (*i.e.*, manually configured by the administrator) between a domain manager and the constrained device. This key is used in HIP-DEX or DTLS in order to authenticate with the domain manager and gain access to the network.

Alternatively, Bergmann *et al.* [24] propose a bootstrapping solution with CoAP. First, the starting node discovers another node that can assist in the bootstrapping. This helping node serves to distribute a temporary secret and establish an association based on DTLS with the pre-shared key (DTLS-PSK). This security association is used to obtain a final session key. With this final session key a new DTLS session can be established. The critical part of the solution lies in the fact that the temporary shared secret is sent in the clear with no protection. Again, the issue with this approach is that the envisioned scenario is only valid for small scale deployments, such as home automation systems.

Korhonen [112] adapts the 3GPP 's *Generic Bootstrapping Architecture* (GBA) [93] to fit in IoT networks. The solution maps the protocols used in GBA, such HTTP and TLS, to its IoT counterparts, CoAP and DTLS, respectively. To simplify the bootstrapping process, a lightweight GBA bootstrapping architecture is proposed. This architecture assumes some pre-configuration of symmetric keys and renders the AAA server unnecessary. As a consequence, the solution is devoid of the AAA's architecture scalability. Moreover, it does not use EAP and it only allows one authentication mechanism, relegating the usability to small/medium scale deployments, such as residential networks.

The *Open Mobile Alliance* (OMA) [158] defines a protocol for managing IoT infrastructures called *OMA Lightweight Machine to Machine* (OMA LWM2M). Among other aspects, bootstrapping is defined using CoAP and DTLS. The architecture consists of three entities, *LWM2M Client*, *LWM2M Server* and *Bootstrap Server*. It is specified that the Client and the Server share credentials with the Bootstrap Server in order to authenticate, but it is not specified how the credentials are configured.

The *IPSO Alliance* [189] defines the use of CoAP as application protocol, using DTLS to protect the devices with sensible resources such as actuators unless other underlying security mechanism are used. No further considerations are added to the security and bootstrapping landscape.

Finally, the W3C [4] considers the case of security on the smart object bootstrapping outside its scope, proposing the adoption of other security implementations such as the *ARM Trust Zone* [18], although there is a proposal aimed at using of GBA [86].

Thus, these solutions do not support large scale deployments, specially for the lack of identity federation support. However, we envisage the necessity of scalable systems

where smart objects from different vendors and organizations can interoperate in large scale scenarios, for example, smart cities.

3.3 The Bootstrapping Service: CoAP-EAP

Our bootstrapping service for IoT rests on three main technologies: CoAP, EAP and AAA. To make this service possible, we have designed a new EAP lower-layer based on CoAP, so it is used to steer an EAP authentication between the smart object and the controller. In turn, the controller interacts with a backend AAA infrastructure to complete the authentication and authorization steps required in the bootstrapping.

As a consequence of the *bootstrapping phase*, fresh cryptographic material is generated and shared between the smart object and the controller to dynamically establish a security association between them. Thus, the smart object will automatically join the controller's security domain, and the controller will become a trusted third party for the smart object. In this work, we present two examples to build this security association (although other options could be considered in the future): either by integrity protecting (encryption will be considered in the future) CoAP messages at application level with a new CoAP option, named *AUTH* (*AUTH-based protection*), or by establishing a DTLS security association (*DTLS-based protection*).

After the bootstrapping, during the *post-bootstrapping phase*, the smart object is able to access other services in the security domain. These services can be provided by other smart objects or entities, such as a border router to access the Internet service. It is worth noting that our solution offers the framework and the cryptographic material to be used in the post-bootstrapping, although the operation in this phase is considered as future work. For example, after the bootstrapping, the controller may act as a key distribution center as is specified in [76] or as an authorization server, as specified in [183].

By defining a bootstrapping service with these technologies, we propose a solution with the following features:

- *Constrained and low-overhead*. CoAP is designed for communications among smart objects in constrained networks. Moreover, we assume that the smart object already ships a CoAP implementation to support other services in IoT networks, so we can re-use the source code for the bootstrapping service.

- *Interoperability*. The solution is based on three well-known standards, which promotes interoperability and easy deployment. The influence that CoAP has on constrained

devices and their use in IoT environments as an application protocol for smart object management benefits interoperability.

- *Security and well-known key distribution and management*. The use of EAP and its associated key management process and the guidelines for AAA key management defined in [88] provides a mature framework for key management.
- *Flexibility*. The use of EAP and AAA provides flexibility in the authentication and authorization processes, so they can be easily adapted to the needs of IoT networks.
- *Scalability and large scale deployment*. AAA framework is already deployed to support millions of users nowadays, for example in 3G networks.
- *Federation support*. AAA provides federated authentication and authorization by design.

The constrained devices that will be able to benefit from this solution will be devices of classes 1 and 2 as described in [33]. Class 0 devices are not considered a target of this solution because of their constraints in memory and processing capabilities as they are not expected to have the resources required to communicate directly with the Internet in a secure manner.

Below, we describe the details of the architecture of our bootstrapping service and how the entities involved are mapped to the EAP-KMF and the AAA framework we have described in Chapter 2.

3.3.1 CoAP as EAP Lower-Layer

One of the first questions that we should answer is the suitability of CoAP as EAP lower-layer. In particular, the requirements for a correct EAP lower-layer are specified in [10]. We can affirm that CoAP can be used as EAP lower-layer by contrasting its capabilities with these requirements:

Unreliable transport. Although EAP does not assume that lower layers are reliable, CoAP provides reliability by means of Confirmable messages. This implies that retransmission timers at EAP level can be stopped for simplification, as recommend in [10]; *Lower layer error detection*. EAP assumes the lower layer has mechanism of error detection. CoAP is performed on top of UDP which already provides a checksum over the whole payload, where CoAP is transported; *Lower layer security*. EAP does not require lower layers to provide security services. CoAP exchanges can be performed without security; *Minimum MTU*. EAP requires a EAP lower-layer with a MTU size of 1020 octets or greater. CoAP assumes an upper bound value of 1024 octets in the payload, where EAP will be transported;

Possible duplication. EAP does not require handling duplication of packets. Even so, CoAP provides a Message-ID for deduplication, which does not harm the EAP authentication process; *Ordering guarantees*. EAP requires the lower-layer to preserve the ordering. CoAP allows us to preserve this ordering by using the Message-ID values. Our bootstrapping service uses this field for that purpose, as described in Section 3.3.3.

As observed, CoAP is able to cover each of the requirements and our bootstrapping service can safely rely on CoAP as a transport for EAP.

3.3.2 Proposed Architecture

To handle the EAP authentication involved during the bootstrapping service, we have designed a new EAP lower-layer based on CoAP. Basically the idea consists of transporting EAP packets in the payload of the CoAP messages involved during the service execution. In general, the Smart Object performs the *CoAP server role* and the Controller the *CoAP client role*. This decision is further explained in Section 3.3.6. To save system resources, it is assumed that the Smart Object will have only a single ongoing bootstrapping exchange and will not process simultaneous EAP authentications in parallel with the same Controller.

Figure 3.1 shows the architecture of our bootstrapping service using CoAP as a transport for EAP packets between the Smart Object and the Controller. We assume that a Controller manages a security domain and, therefore, the bootstrapping process. In this way, any smart object or entity wanting to join the security domain will have to engage with the Controller.

The Smart Object acts as *end user* in the AAA framework, and it will act as EAP peer and perform the client-side of a particular EAP method. The Controller acts as the EAP authenticator for the bootstrapping service and acts as the EAP authenticator (typically in pass-through mode for big scale deployments) and ships a AAA client (RADIUS or Diameter) to interact with the backend AAA infrastructure. The EAP server is typically placed in the IdP 's AAA server where the Smart Object is registered. Several intermediate AAA proxies can be placed between the Controller and the IdP 's AAA server, especially in federated environments (although it will depend on the specific deployment) [212].

After finishing the bootstrapping service, the Controller becomes a trusted third party in the security domain for the Smart Object so that it can interact in a secure fashion with the rest of the entities within the domain.

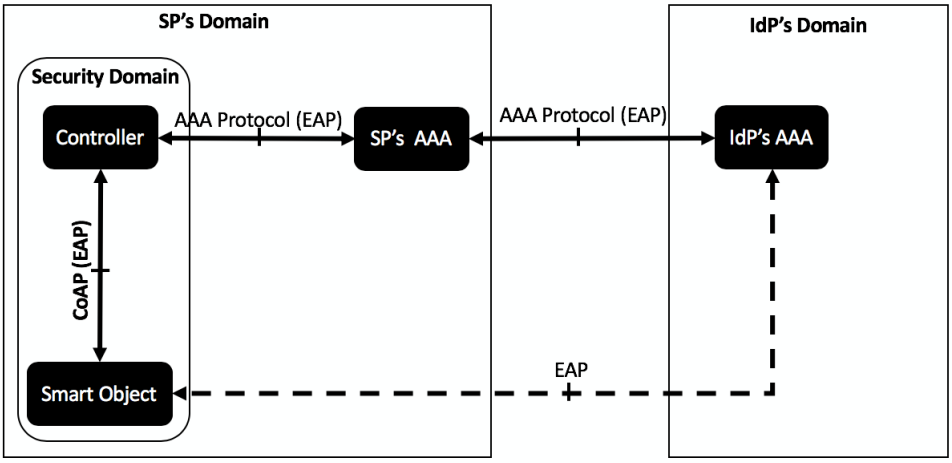


Fig. 3.1 The constrained application protocol (CoAP)-EAP architecture.

3.3.3 General Operation Flow

In order to run the bootstrapping service, it is necessary to define a *Uniform Resource Identifier* (URI) so that any endpoint can refer to the service using that value. In particular we defined the URI as */boot* for our bootstrapping service.

In this manner, when the Smart Object, acting as CoAP client, wants to start the bootstrapping service, it sends a Confirmable *POST /boot* request to the Controller, which acts as CoAP server in this first exchange (step **1**). Typically, the Controller will answer back with a response message (step **1'**) saying the service is available (CoAP Response Code *2.04*). However, it may omit it and proceed with the following exchange to save this message in the link. Then, the Controller, as CoAP client for the rest of the exchange, immediately sends a Confirmable *POST /boot* request to the Smart Object as CoAP server from this point on (step **2**). The Smart Object indicates the creation of a resource for the bootstrapping service to the Controller. The message carries a nonce (*nonce-c*), which will be used for the generation of fresh cryptographic material after the bootstrapping execution; and a Token with a value chosen randomly. This value is used as the session identifier and kept during the whole authentication. The smart object assigns an identifier to a resource (value 5 in the example) and answers with an ACK that carries a piggybacked response with a new nonce (*nonce-s*), also used for the same purpose than *nonce-c*, and the Token (step **3**).

The *Message-ID* (MID) values in the requests sent by the Controller are generated *randomly*, as suggested in the CoAP standard. The Controller selects a new Message-ID value each time a new request is sent to the Smart Object, until the bootstrapping service finishes. Moreover, the Controller stores the last Message-ID sent until correctly receiving the corresponding ACK. The Smart Object keeps track of the last received Message-ID to

identify retransmissions, and the previous Message-IDs during the current bootstrapping to identify old messages. In general, a request is considered valid in terms of the Message-ID if either this value matches the last value received, which means a retransmission of the last response is required, or the arrival of a new Message-ID, which therefore represents a new message. If these rules do not apply (*i.e.*, an old Message-ID has been received), the Smart Object silently discards the request. This is possible because the bootstrapping service is designed as lockstep: the Controller will not send a new request until receiving the corresponding response. When the current bootstrapping exchange finishes successfully, the smart object can free the tracked Message-IDs, except for the last received Message-ID at the end of the bootstrapping, just in case a retransmission is required.

After this initial handshake, the EAP authentication starts. Figure 3.2 shows an example of exchange using a generic EAP method (EAP-X) and pass-through mode (IdP 's AAA server intervenes in the EAP authentication). Nevertheless, the number of messages will depend on the EAP method used. The Controller will use the POST method to send EAP requests to the resource created in the Smart Object. Then, the Smart Object sends an ACK with a piggybacked response to carry the EAP responses to the Controller (steps **4**–**16**). This corresponds with *phases 1a and 1b* in the EAP KMF (RFC 5247 [12]).

Specifically, the Controller first requests the Smart Object's identity (*EAP Req/Id*) (e.g., *smartobject@domain.net*) (step **4**). Then it sends its identity (*EAP Res/Id*) in an ACK message (step **5**). The Controller uses the information in the identity to route the EAP Response/Id message to the IdP 's AAA. Without loss of generality, our example is based on RADIUS as AAA protocol. In Figure 3.2 we can observe the RADIUS *Access-Request* message containing the EAP Res/Id (step **6**). To inform the IdP 's AAA that this EAP authentication is for a bootstrapping purpose, the Controller includes the *Service-Type* attribute with a new value: *Bootstrapping*. This is required so that the IdP 's AAA can apply authorization decisions adapted to the bootstrapping service.

Based on this information, the IdP 's AAA server selects the proper EAP method for authentication. In our example, this will be the EAP-PSK method, which implies 4 messages (steps **7**–**14**). EAP requests of EAP-PSK are transported to the Controller from the IdP 's AAA server by using *Access-Challenge* messages and responses are received from the Controller by means of *Access-Request*.

At the end of the message exchanges, if everything has gone as expected, the Controller receives cryptographic material (*i.e.*, MSK) from the IdP 's AAA server along with the *EAP Success* message in a *Access-Accept* message. In turn, the Smart Object will be able to generate the same keying material as defined by the EAP KMF specification. In addition to this information, the Controller may receive authorization information from the AAA

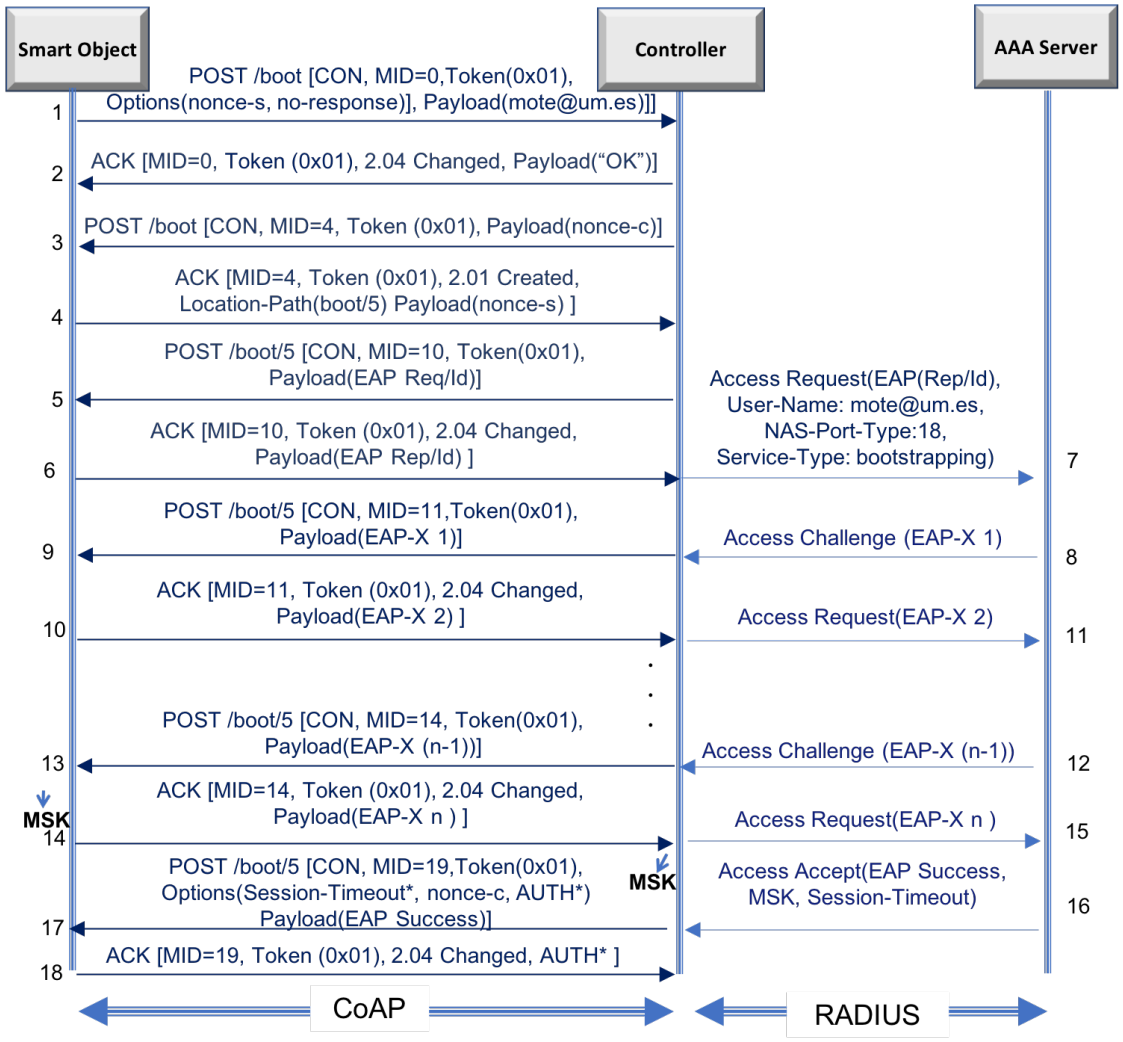


Fig. 3.2 CoAP-EAP bootstrapping service flow (using a generic of EAP method (EAP-X)).

infrastructure, for example, the session lifetime (*Session-Timeout* attribute in RADIUS) related with this bootstrapping process. Part of this information may be passed to the Smart Object through, for example, a generic CoAP option named *Authorization Option* (step **16**). All this information is stored in the so-called *bootstrapping state* (see Section 3.3.5).

At this point, the Smart Object and the Controller share cryptographic material (MSK) to establish a security association enabling the signaling to be protected between both entities for further service requests. This corresponds to *phase 2a* in the EAP KMF [12]. In this chapter, we show two examples of usage of this keying material, though other alternatives may be considered in the future: first, the integrity protection at application level of CoAP messages by means of a new AUTH option; second, the establishment of a DTLS security association. In the following sections, we present some details about these alternatives, describe how to manage the bootstrapping state and provide additional considerations.

3.3.4 Bootstrapping Security Associations

Key Hierarchy Design

The first step to protect services and messages after the bootstrapping is to design a key hierarchy. If AUTH-based protection is used (see Section 3.3.4) a new key named CoAP _PSK is derived. In contrast, if DTLS is used, a DTLS_PSK is derived (see Section 3.3.4).

Additionally, we have considered the derivation of an *Application-Specific Root Key* (ASRK), which is used as a root of an additional key hierarchy. By using the ASRK, the Controller can acts as key distribution center for the recently bootstrapped Smart Object. For example, this key material will allow the Smart Object, if properly authorized, to securely access to the services offered by the Controller, other bootstrapped Smart Objects or entities within the security domain, which also have a security association with the Controller (the trusted third party for all of them). How this key is used will depend on the interactions expected in the security domain and the services accessed during post-bootstrapping. In fact, it could be used to distribute key material to layer 2 or layer 3 security. In any case, it is left out of this work and will be covered as part of future work.

CoAP Message Protection at Application Level: AUTH Option

Once the Controller has received the MSK from the AAA server, it can derive the CoAP_PSK. This key is used to generate the values for the new defined AUTH option, which contains a *Message Authentication Code* (MAC) for integrity protection over the entire CoAP message. The reason for using this option is that it has been proved that running a DTLS handshake can be really costly in terms of time in IEEE 802.15.4 networks [207]. Thus, certain applications

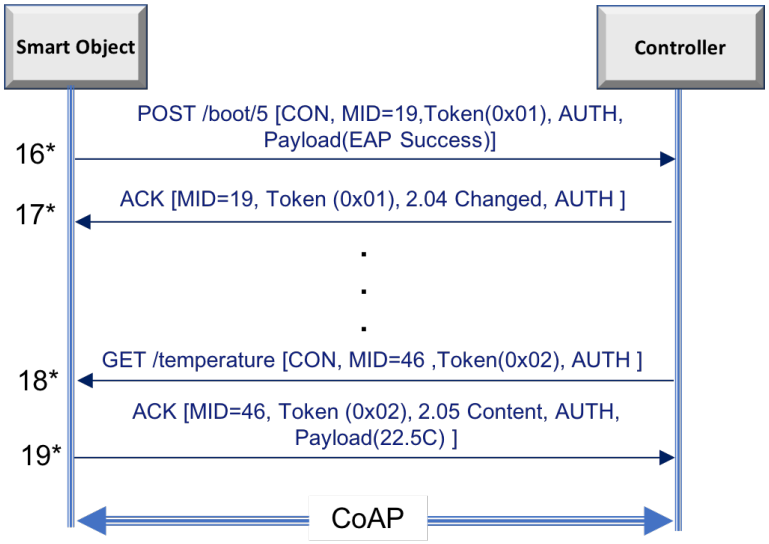


Fig. 3.3 AUTH-based protection example.

may prefer not to perform the DTLS handshake. In fact, other types of (non-IEEE 802.15.4) networks considered in IoT, generically referred to as LPWAN [134] are generally low power and low-throughput [213] and, therefore, saving bits in the link is a benefit.

The first time this option appears is during the bootstrapping phase in the message that conveys the EAP Success (step **16'**). The corresponding ACK message (step **17'**) from the Smart Object, which closes the bootstrapping phase, also includes an AUTH option. By verifying the MAC, one endpoint can know whether the other endpoint owns the CoAP _PSK.

Since AUTH option is a new type of protection, a new port (to be assigned by IANA) and a new URI scheme identifier (e.g., "coapa") should be allocated. Thus, this last exchange will go through this new port. From this point, any message related with the bootstrapping service (e.g., to remove the bootstrapping state as explained in Section 3.3.5) will include the AUTH option.

Any other service (e.g., to obtain the temperature from the smart object), implying the communication between the Smart Object and the Controller, can also use the AUTH option (steps **18'** and **19'**). Nevertheless, this does not preclude the use of DTLS, as we will see in Section 3.3.4 with the derivation of the DTLS_PSK. In fact, the use of either DTLS or AUTH will depend on the service URI, "coaps" or "coapa", respectively. The use of different URI s to indicate the protection of the exchange are exclusive. The reason is that it is not useful to provide integrity with the AUTH option and ciphering with DTLS, since DTLS already offers the possibility of using integrity and ciphering. Hence, the use of the AUTH option is used when a very simple and less taxing approach is needed.

COAP_PSK is a 16-byte length key which is computed using AES-CMAC-PRF-128 [195] as KDF, which, in turn, uses AES-CMAC-128 [196]. Both primitives use AES-128 [28] as building block since it is widely used in constrained devices.

$$COAP\_PSK = KDF(MSK, \text{“}IETF\_COAP\_PSK\text{”} || nonce-c || nonce-s, 64, length) \tag{3.1}$$

where

The *AES-CMAC-PRF-128* is defined in [195]. This function uses AES-CMAC-128 as a building block; The *MSK* is exported by the EAP method; “IETF_COAP_PSK” is the *American Standard Code for Information Interchange* (ASCII) code representation of the non-NULL terminated string (excluding the double quotes around it). This value is concatenated with the value of the nonces exchanged; *64* is the length of the MSK; *length* is the length of the label “IETF_COAP_PSK” (13 bytes) plus the two nonces; *nonce-c* is a random value sent from the Controller to the Smart Object; *nonce-s* is a random value sent from the Smart Object to the Controller.

To calculate the MAC value, an endpoint inserts the AUTH option and sets its value to 16 bytes with zero. Then, it applies the function AES-CMAC-128 to generate the AUTH Option value over the entire message as follows:

$$AUTH\ Option\ value = AES-CMAC-128(COAP\_PSK, message, length) \tag{3.2}$$

where

COAP_PSK is the key derived in Equation (3.1); *message* is the CoAP message including the AUTH option filled with zeros; *length* is the length of the CoAP message including the AUTH option.

It is worth mentioning that while we propose the use of an AUTH Option just as an example to show the protection of CoAP messages at application level, there are additional works, such as [214], that define additional CoAP Options to cope with protection at application level. In parallel, Object Security for CoAP (OSCOAP) has been considered [186] to protect CoAP messages with COSE [181] objects.

CoAP Message Protection with DTLS

DTLS is the standard protocol used to protect CoAP messages by default. If the Controller wants to use DTLS for protecting messages it will not send the AUTH option (see Figure

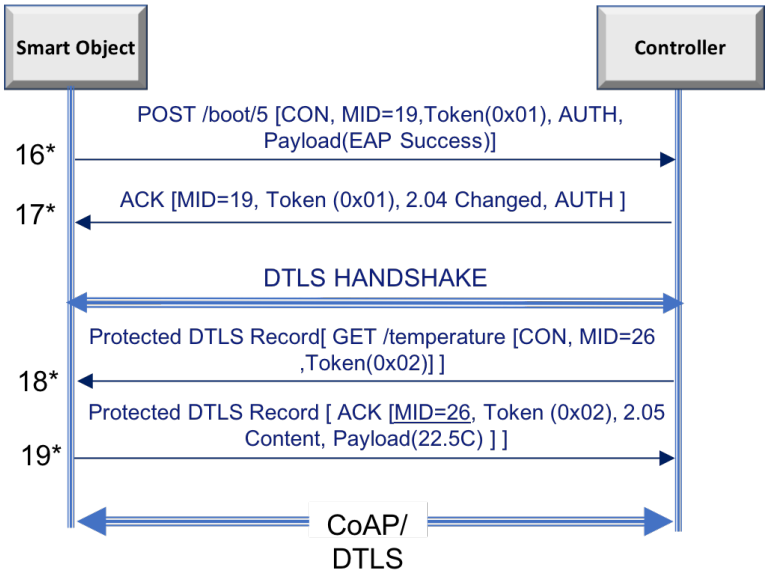


Fig. 3.4 Protecting post-boostrapping with datagram transport layer security (DTLS).

3.4) (step **16'**). In this case, it derives a DTLS_PSK from the MSK (the Smart Object will do the same) and starts a DTLS negotiation over port 5684. Thus, the Smart Object and the Controller do not consider the bootstrapping to be complete until the DTLS handshake finishes successfully.

Now, messages related with the bootstrapping service that may be exchanged from this point (*i.e.*, Confirmable DELETE) are protected by DTLS. Any other service can still use DTLS (steps **18'** and **19'**) if the URI scheme for this service contains "coaps" (or even decide to use AUTH option deriving the CoAP _PSK if the URI scheme contains "coapa").

The DTLS_PSK will also have 16 byte length and will be derived as follows:

$$DTLS\_PSK = KDF(MSK, \text{“}IETF\_DTLS\_PSK\text{”} || nonce-c || nonce-s, 64, length) \tag{3.3}$$

where

MSK is exported by the EAP method; *"IETF_DTLS_PSK"* is the ASCII code representation of the non-NULL terminated string (excluding the double quotes around it); *64* is the length of the MSK; *length* is the length of the label "IETF_DTLS_PSK" (13 bytes) plus the two nonces; *nonce-c* is a random value sent from the Controller to the Smart Object; *nonce-s* is a random value sent from the Smart Object to the Controller.

As mentioned in [54], a PSK identity is needed. We consider the use of the Token value chosen during the EAP authentication as PSK identity.

3.3.5 Bootstrapping State Definition and Management

The *bootstrapping state* is a set of parameters resulting from the bootstrapping process described in Section 3.3.3. This state is maintained by the Smart Object and the Controller. The bootstrapping state comprises the following values: both entities' IPv6 addresses; authorization information related with the bootstrapping service that came form the AAA infrastructure, in particular, the supported cryptosuite, capabilities (e.g., AUTH-based or DTLS-based protection) of the Smart Object and the lifetime associated to the state; list of keys exported and derived from the bootstrapping procedure (COAP_PSK, DTLS_PSK and ASRK) and a resource identifier generated in the Smart Object after receiving a Confirmable POST /boot request.

In general, if the lifetime of the bootstrapping state expires at both endpoints it will be automatically removed, along with the associated resource. Nevertheless, the Smart Object or the Controller can explicitly signal a desire to remove the bootstrapping state.

On the one hand, the Smart Object may also request the Controller to abandon the security domain and, thus, to delete bootstrapping state. This is useful because the Smart Object may desire to notify the Controller that it should stop sending accounting information to the AAA infrastructure since, for example, it is leaving the security domain. In a similar way as the Smart Object signals that it wants to start a bootstrapping by sending a Confirmable POST message to the URI of the bootstrapping service (*POST /boot*), the Smart Object can signal the Controller its desire to leave the security domain by the following URI-Path: */boot?del=X*. The part */boot* indicates that the Smart Object requires the Controller to perform some action related to the bootstrapping. This action is defined by the part *?del=X*, which signals to the Controller that the Smart Object desires to leave the security domain, where *X* is the resource identifier to be erased.

For example, Figure 3.5 shows the Smart Object sends a protected *Confirmable POST /boot?del=5* (step **20**) to trigger the Controller to remove resource identifier 5. When the Controller receives this request, it looks for an existing bootstrapping state that matches the identifier. If the Controller has the bootstrapping state, it sends a Confirmable DELETE request (step **21**) to the Smart Object to erase the resource with the assigned identifier (e.g., *"boot/5"*). Typically, the Controller can also send the response associated to the POST but the Smart Object considers that receiving the DELETE directly is sufficient. In fact, upon receiving the Confirmable DELETE request, the Smart Object sends a protected ACK with piggy-backed response (step **22**) with a response code to confirm the bootstrapping state

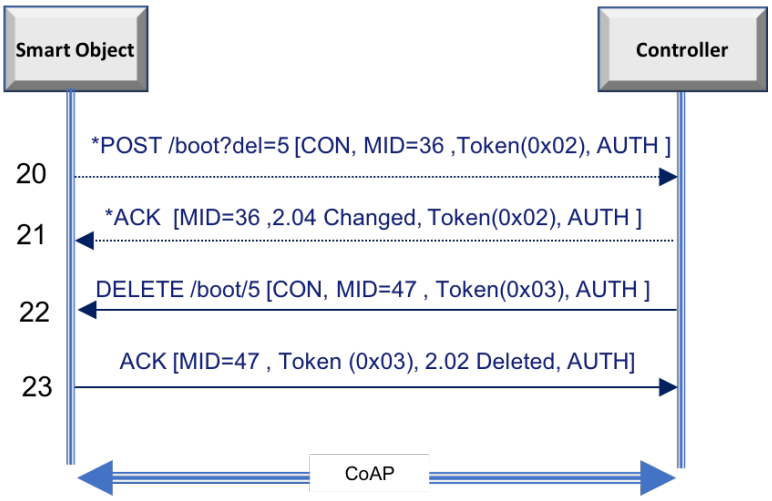


Fig. 3.5 Example of deleting bootstrapping state (with AUTH-based protection).

is going to be removed (*2.02*). If the Controller already removed the state (e.g., it already expired), it sends an unprotected (there is no bootstrapping state) ACK message with code *4.04 (Not Found)*. The Smart Object may not trust this unprotected message and insist on sending POST (e.g., 3 times) to see if it receives a protected Confirmable DELETE request. If the Smart Object does not finally receive this request, it may consider that the Controller has already removed the state. On the other hand, if the Controller wants to delete the bootstrapping state, it only has to start the Confirmable DELETE request/ACK exchange.

Finally, the Smart Object may want to renew the bootstrapping state (refreshing cryptographic material, lifetime, *etc.*) before it expires. This means a new EAP authentication, as we have described in Section 3.3.3. This new bootstrapping exchange needs to finish before the current bootstrapping state expires. If the new bootstrapping process finishes successfully, the current bootstrapping state is replaced with the new one.

3.3.6 Additional Considerations

CoAP Role Selection

With the exception of the *POST /boot* message sent by the Smart Object to notify the Controller the beginning of a bootstrapping process, or the *POST /boot?del=X* to trigger the removal of resource *X*, the rest of exchanges assume that the CoAP server is the constrained Smart Object and the CoAP client is the Controller. The main reason for this role choice is, as suggested by [114], to simplify the Smart Object implementation, assuming this will be the most constrained entity. Additionally, in EAP, the authenticator sends EAP requests from the peer, which returns an EAP response. The authenticator carries the burden of retransmissions

including additional states for this purpose in the EAP state machine [205]. Similarly, in CoAP, the CoAP client sends CoAP requests, and the CoAP server just answers back with a response. With our design choice, a CoAP request carries an EAP request and a CoAP response transports an EAP response, avoiding the odd case where EAP requests go into CoAP responses and vice-versa, which complicates the overall design.

Discovering the Controller

One aspect that has not been discussed until now is how the Smart Object discovers the Controller. Or, in other words, how the Smart Object knows which entity supports the bootstrapping service in a particular security domain. Actually, this process lies outside the bootstrapping service, since there are already mechanisms for that process in CoAP. For example, the CoRE Resource Directory [191] provides the framework to discover services.

In particular, it describes an entity called *Resource Directory*, which can register the bootstrapping service (*rt = "bootstrapping"*). The access to this information should be public and not protected since it will be the first service to be used for Smart Objects that want to join the security domain. The Resource Discover will return a "coap" URI schema (e.g., $coap://[IP-Controller]/boot$) for our bootstrapping service, meaning the initial exchanges goes to the unprotected CoAP port.

Trusting the Controller

One question that may remain is how the Smart Object can trust the Controller. Basically this is explained and discussed in the EAP KMF (Sections 2.3 and 3.1 Part b) [12]. In short, the trust is based on the fact that the controller receives and uses (proof of possession) the MSK (in reality a key derived from the MSK such as CoAP _PSK or DTLS_PSK) to establish a security association after a successful authentication. Since the IdP 's AAA server provides this MSK to the Controller and the Smart Object trusts the IdP 's AAA server, if the AUTH option is verified correctly it means the controller obtained the MSK from a trusted entity (the IdP 's AAA server) and derived the CoAP _PSK. Alternatively, if the DTLS handshake is used and finishes correctly, it means the Controller was able to derive the DTLS_PSK from the MSK.

Additionally, the Controller needs to be trusted by the AAA infrastructure. This is typically done by the SP administrator configuring the Controller with a credential (e.g., a shared secret) that can be verified by the SP's AAA server. Moreover, the SP's AAA server is configured to register the Controller as a trusted entity. In the same way, SP's AAA server

is trusted by the IdP 's AAA server. Thus, the Controller is trusted by the AAA infrastructure for a principle of trust transitivity, which is the common model in AAA infrastructures [44].

Authorization Aspects

Authorization is also an important part of the bootstrapping process. The authorization information related to the bootstrapping service (*bootstrapping authorization information*) determines, for example, whether the Smart Object can join the security domain and under what conditions. Since our bootstrapping solution is based on AAA framework, the authorization information is carried to the Controller in the form of AAA attributes. RADIUS and Diameter define a myriad of attributes (e.g., NAS-Filter-Rules, Framed-MTU, Session-Timeout, *etc.*) to carry this information. For more fine-grained authorization, [89] has recently specified how to transport SAML in RADIUS attributes, though more constrained authorization information (in term of message size) could be expressed in *JavaScript Object Notation* (JSON) [39] or CBOR [34] format. However nothing has yet been defined in this regard.

We consider that the set of attributes already defined in RADIUS or Diameter could be a good starting point for a basic and correct authorization of the bootstrapping service. In particular, we have paid attention to the lifetime associated to the bootstrapping state or the service type. Nevertheless, if other types of information are required (for example, certain domains may require a hardware profile of the Smart Object to take an authorization decision regarding the Smart Object, even though it is successfully authenticated), this may require the definition of new IoT-based AAA extensions, such as [194]. Nevertheless, the AAA framework is prepared to be easily extended so giving the required flexibility to provide any application-specific authorization information needed for IoT applications.

Additional authorization information may be required *after* the bootstrapping for the operation in the security domain. For example, this is required for accessing services provided by other Smart Objects, the Controller, border routers, *etc.*, in the security domain. This information is also application specific and considered as part of future research. Nevertheless, the IETF ACE WG is investigating solutions for this post-bootstrapping authorization process.

Finally, how the Controller will handle any authorization information will depend on the local security policies and the activity of the Smart Object in the security domain. Additionally, if some authorization related information needs to be carried to the Smart Object we will use our CoAP-based EAP lower-layer for the transport. The general framework to support this is using CoAP options containing the required information. For example, an *Authorization* Option may be included in the CoAP message to notify the Smart Object with the bootstrapping authorization lifetime value, if this value is different from the default,

which is 8 hours following the hint in [12]. Knowing the session lifetime will allow the Smart Object to arrange a new bootstrapping before the existing bootstrapping state expires.

Cryptographic Suite and Protection Selection

In order to simplify the bootstrapping service, we assume a default cryptographic suite based on *Advanced Encryption Standard* (AES) algorithm. We assume AES-CMAC-PRF-128 as the default KDF (see Section 3.3.4); AES-CMAC-128 to generate the new AUTH option (Section 3.3.4); and, when encryption and integrity protection are required at CoAP application level, we assume *AES in Counter with CBC-MAC (CCM) Mode* (AES-CCM) [211]. In any case, some sort of negotiation is important to support cryptographic agility [138].

In our model, the controller is the entity which decides the cryptographic suite that must be used in the security domain. Thus, the information the controller needs to know about the Smart Object is the supported cryptosuite. Since our solution is AAA-centric, the IdP 's AAA, where the smart object is registered, can provide this information to the Controller during the bootstrapping, so that it can finally decide what option to choose. Thus, this information is not sent over the constrained network reducing the CoAP message size of the bootstrapping service. Additionally, the IdP 's AAA can inform if the smart object supports AUTH-based protection, DTLS-based protection or both.

In contrast, the AAA protocol needs to carry this information. However, there are no standard attributes for this. It means defining several new attributes: *Encryption-Type*, *KDF-Type*, *Protection-Type* and *MAC-Type*. These attributes would carry one octet specifying the algorithm (for example, using the value in [218]) for ciphering; the KDF to derive cryptographic material; the function to generate the AUTH option value, and the type of protection supported by the smart object, respectively.

The IdP 's AAA server may include an attribute of this type per each supported algorithm when sending the EAP Success to the Controller. For simplicity, if these attributes are not included, the Controller will assume that the default values are those supported by the smart object and that both AUTH-based protection or DTLS-based are supported.

Finally, the smart object needs to be informed about the decision as well. This can be done in the last POST message containing the EAP Success (step **16**) through a new option *Cryptosuite* that contains 3 bytes (3-tuple) with: encryption, KDF and MAC algorithms. Nevertheless, to avoid increasing the message size, if this option is not included it is assumed that the default values will be the chosen ones. The use of AUTH-based or DTLS-based protection is determined because the Smart Object sees a message over the DTLS port and

protected with DTLS or a message with AUTH option over the yet-to-be defined port for AUTH-based protection.

Other Security Considerations

On the one hand, our bootstrapping service makes use of EAP and AAA and associated key management. In terms of security, the solution does not add anything different than existing deployments of EAP and AAA do. In this sense, it follows the *EAP Key Management Framework* [12] and the *Guidance for Authentication, Authorization, and Accounting (AAA) Key Management* [88].

On the other hand, the AES-based cryptosuite selected as default is well-known and already deployed nowadays in IoT devices [136]. It is worth noting that the cryptosuite negotiation is for the operation at bootstrapping service level. In other words, the DTLS negotiation can select a different set of algorithms.

Nevertheless, we should clarify how we avoid any downgrading attack during the cryptosuite selection procedure shown in Section 3.3.6. This attack happens when an attacker removes some of the algorithms to limit the set of valid cryptographic algorithms to those more favorable to the attacker. We assume that AAA infrastructure is trusted; therefore the path between the Smart Object and the Controller is where the attack may happen. Our mechanism to detect this attack is simple. If protection at level application is enabled (AUTH option) the last POST message and corresponding ACK (Figure 3.2—steps **16** and **17**) are integrity protected so any modification (removal or modification of the algorithms specified in the POST message) will be detected by both entities. On the contrary, if DTLS is used, we can observe in Figure 3.4 that the last POST message and corresponding ACK are not protected. This is why the Smart object and the Controller must end the DTLS negotiation before considering the bootstrapping complete. In reality, the KDF is the only function which needs to be agreed in this case, since DTLS already performs a cryptosuite negotiation. Thus, if an attacker has modified the KDF algorithm, the DTLS _KEY derived by the Controller and the Smart object will be different and DTLS will fail, so that the attack is detected.

3.4 Experimental Results

In this section we present the testbed we have defined to evaluate the performance of the CoAP-EAP implementation done for this purpose. To complete the evaluation and analysis we compare CoAP-EAP with PANA-based solutions, which have a similar purpose but the EAP lower-layer is based on PANA. As a representative of PANA-based solutions, we have chosen PANATIKI since it can be considered as a best case for these type of solutions. The

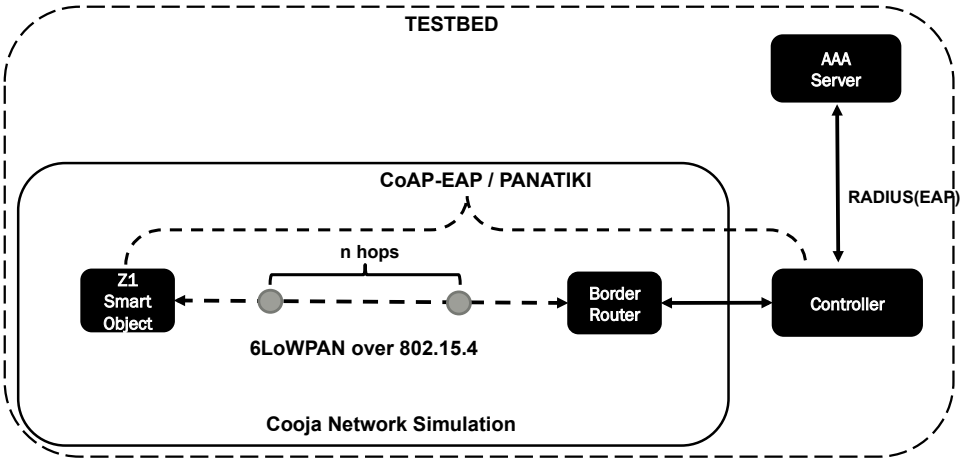


Fig. 3.6 Testbed for the evaluation.

reason is that PANATIKI is a design and implementation of PANA optimized for constrained devices.

In particular, we have compared CoAP-EAP with AUTH-based protection against PANATIKI. The reason is that, after bootstrapping, running DTLS can be considered as the same task in both alternatives, thus not adding value to the comparison. For the performance of DTLS, the interested reader is referred to [207]. By including AUTH option we show the worst case in terms of the operation of our CoAP-based EAP lower-layer (it has to process the AUTH option) against the best case of PANA-based solutions, which is PANATIKI. Even so, there is still some room for improvement, as we will analyze in the next sections.

3.4.1 Experimental Setup

Figure 3.6 shows the testbed we have prepared for our performance evaluation. For the testbed we use the Cooja Network Simulator with Contiki OS in its version 2.7 [146]. The Smart Objects used for this testbed are the Zolertia Z1 with 92 kB of nominal ROM when compiled with 20-bit architecture support, and 8 kB of RAM. The compiler is *msp430-gcc version 4.7.2*.

There is an entity in Cooja, the *RPL border router*, that enables the communication between the Cooja Network and the outside physical network where the Controller is located. Between the border router and the Smart Object thre can be 1 hop (direct link), or other smart objects between the Controller and the Smart Object performing the bootstrapping. The idea of having several hops is to observe the behavior of the bootstrapping time when intermediate Smart Objects are acting as IP-forwarders.

Table 3.1 Testbed PC.

CPU	Intel(R) CoRE (TM) i5-2400 CPU @ 3.10GH
RAM	4GiB DIMM DDR3 Synchronous 1333 MHz
O.S.	Ubuntu Server 12.04.5 LTS - 32 bits
Kernel	3.13.0-32-generic

For the tests, we used a 4-byte length Token in CoAP-EAP, which provides a 32-bit for the session identifier, as in the case of PANA. Following the recommendations in [3], we have performed the simulations in Cooja with a randomly generated seed to automate the process of running the simulations. We have used the default parameters in Contiki for the MAC layer and RDC. In particular, the parameters for the simulation includes the *contikimac_driver* for *Radio Duty Cycling* (RDC) and *csma_driver* for *Carrier Sense Multiple Access* (CSMA) with default values. Due to the length of the PANA messages, we have had to set the parameter UIP_CONF_BUFFER_SIZE to 250 in the Contiki OS so that PANATIKI works correctly (otherwise no PANA authentication was completed). The same parameter is kept in CoAP-EAP for fair comparison.

In term of the software packages, on the one hand, we have used the PANA agent (PAA) implementation of OpenPANA [131] for the Controller in PANATIKI [1]. With the purpose of making a fair comparison, we have implemented the Controller-side of our CoAP-EAP bootstrapping service using the same PAA implementation but replacing PANA with our CoAP-based EAP lower-layer source code. To develop this new EAP lower-layer, we used *cantcoap* (https://github.com/staropram/cantcoap) as CoAP library. Although we tested several CoAP libraries: libcoap (http://libcoap.net), erbium (http://people.inf.ethz.ch/mkovatsc/erbium.php) and cantcoap, we decided to go for cantcoap for its simplicity, which gave us greater control over the implementation.

On the other hand, PANATIKI is used to implement the PANA client (PaC) in the smart objects for PANA-based solutions. In CoAP-EAP, we have transformed *cantcoap* from C++ to C version for the compilation in Contiki OS. The same EAP peer state machine of PANATIKI is used in CoAP-EAP. Finally, we have used FreeRADIUS (http://freeradius.org/) version 2.0.2 for the role of AAA Server.

For CoAP-EAP and PANATIKI, the EAP method used to obtain experimental results is EAP-PSK due to its lightweight nature. The EAP-PSK keys used are 16 bytes long and the EAP identity is 6 bytes long. The implementation of EAP-PSK is common in both cases. In the EAP peer side, EAP-PSK implementation is the one provided in PANATIKI. On the EAP server side, the EAP-PSK implementation is that in FreeRADIUS 2.0.2.

[1] http://sourceforge.net/projects/panatiki/

Table 3.2 Message length.

CoAP-EAP			PANATIKI			% CoAP-EAP Reduction	
Msg.	**LL**	**LL+EAP**	**Msg.**	**LL**	**LL+EAP**	**LL**	LL**+EAP**
POST	13	13	PCI	16	16		
POST(nonce-c)	18	18	PAR	40	40		
ACK(nonce-s)	20	20	PAN	40	40		
POST(Req/Id)	17	22	PAR(Req/Id)	43	48		
ACK(Res/Id)	9	20	PAN(Res/Id)	41	52		
POST(EAP-PSK 1)	17	46	PAR(EAP-PSK 1)	27	56		
ACK(EAP-PSK 2)	9	69	PAR(EAP-PSK 2)	24	84 *		
POST(EAP-PSK 3)	17	76 *	PAR(EAP-PSK 3)	25	84 *		
ACK(EAP-PSK 4)	9	52	PAR(EAP-PSK 4)	25	68		
POST(EAP Success)	36	40	PAR(EAP Success)	84	88 *		
ACK	27	27	PAN	52	52		
Total	192	403		417	628	53,9%	35,8%

LL: lower-layer message length. LL+EAP: lower-layer message length including EAP message length

3.4.2 Performance Evaluation

Message Length

In general, the message length may influence the time the Smart Object takes to process it but, more importantly, the time that takes to send and receive it over the network. This aspect gains relevance in lossy networks where, for example, fragmentation becomes a matter of utmost importance [38]. Currently, the IEEE 802.15.4 defines a MTU of 127 bytes.

Table 3.2 shows the message length (in bytes) for CoAP-EAP and PANATIKI. We show the length of the EAP lower-layer, excluding the length of the EAP message (*LL*) and including the EAP message length (*LL+EAP*). Thus, in the case of CoAP-EAP, *LL* column includes the length of CoAP header (4 bytes), the (variable) list of CoAP options and payload, excepting the length of EAP message itself. As PANATIKI is an implementation of PANA, it follows the standard, so the PANA message length is the same as specified in RFC 5191 [61]. Thus, *LL* column for PANATIKI includes the length of the PANA message excepting the EAP message length. In short, a PANA message includes the PANA header (16 bytes) and a variable length payload. This payload is formed for a list of *Attribute-Value Pair*s (AVPs) (e.g., the PANA message containing the EAP success, *PAR(EAP Success)*, includes a list of 5 ASRK s). Each ASRK has a *Tag-Length-Value* (TLV) format and its length is *8 bytes* plus the length of the content of the ASRK. For the specific list ASRK s carried in each PANA message during a PANA authentication, the interested reader can refer to [61]. Finally, *LL+EAP* column adds the EAP message length to the values in column *LL* in both cases.

As a consequence, all the messages related with CoAP-EAP have a shorter length in comparison with PANATIKI's. In fact, it is worth noting that CoAP is designed with a *short*

fixed-length binary header and compact binary options [190] and PANA was not designed with the constraints of IoT environments in mind. Overall, there is ≈36% reduction in the number of bytes between both alternatives when looking at LL+EAP. Apart from IEEE 802.15.4 networks, we foresee that this will be also important, for example, in low power wide area networks [134] where sending a single byte is usually very costly. If we pay attention just to the CoAP-based EAP lower-layer that we have designed, since the EAP implementation is common in both approaches, the reduction increases about 54%.

We expect additional impacts of our solution as a consequence of the reduction of message length in networks where the bandwidth is really low. In particular, several discussions are going on in the context of the IETF *IPv6 over the TSCH mode of IEEE 802.15.4e* (6TiSCH) WG [209] about the use of CoAP as transport of EAP in this type of networks.

Additionally, avoiding fragmentation is important. PANATIKI surpasses this MTU threshold, 127 bytes MTU in IEEE 802.15.4, in three messages (marked with $*$ in Table 3.2) in contrast with CoAP-EAP, with only one. Although the lower-layer plus EAP message lengths are below 127 bytes, the whole packets including the MAC layer and 6LoWPAN layer, surpasses the MTU. Thus, this is a hint to consider a different EAP method, with shorter messages as well.

Bootstrapping Time

We have defined a scenario with different numbers of hops between the Smart Object and the border router. This allow us to evaluate the performance of each protocol in a more realistic scenario than a simple node connected to a border router.

The tests have been performed with different numbers of hops (n), from 1 to 7 hops between the Smart Object and the border router. We have observed though that 7-hop case has proved to be a scenario without connectivity between the Border Router and the Smart Object being bootstrapped. Three different packet loss ratios (0, 0.1 and 0.2) are used to evaluate the response of both solutions.

From the different tests we have gathered the following information: (1) the bootstrapping median time that both alternatives take to complete a bootstrapping for each number of hops and packet loss ratio (Figure 3.7a–c); and (2) the number of bootstrapping processes that are able to finish (success percentage (%)) in each case (Figure 3.8a–c).

We can observe in Figure 3.7a–c that there is a statistically significant difference between the bootstrapping times in PANATIKI and CoAP-EAP. To obtain these values we performed 200 simulations for each value of n and for each packet loss ratio.

On analyzing the distribution of the data, we noticed that, a skewness test on the data, returned a value greater than 1. According to Jain [99] a proper index of central tendency is the median rather than the mean, since it provides a more significant description.

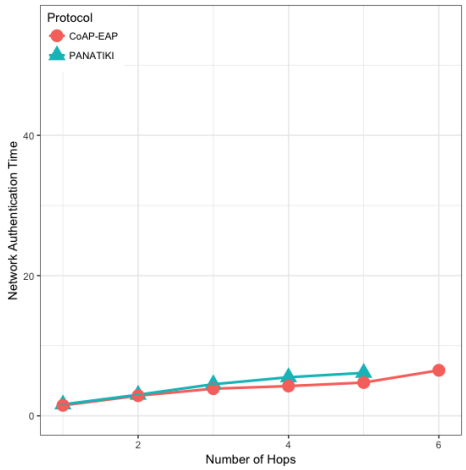
In the case of 0 loss ratio and 1 hop, the median values do not differ greatly. In this case CoAP-EAP takes ≈1.5 s and PANATIKI ≈1.6 s. As expected, this difference, partially due to the reduction in message size of our solution (see Section 3.4.2), increases with the number of hops and packet loss ratio since each intermediate smart object has to forward a message and handle fragmentation of these messages. Since PANA has longer messages, PANATIKI provides a longer bootstrapping time than CoAP-EAP. For example, when packet loss is set to 0.2 loss ratio with 1 hop, the median time to accomplish a bootstrapping in CoAP-EAP ≈5 s and ≈6 s in PANATIKI. As expected, this time clearly increases with the increment of number of hops.

Moreover, when the packet loss ratio increases the possibility of completing a bootstrapping decreases. This is implicitly shown in Figure 3.7a–c. For example, we were not able to complete a bootstrapping process with PANATIKI beyond 5 hops with 0 packet loss ratio; 4 hops when 0.1 packet loss ratio and 3 hops with 0.2 packet loss ratio. However, CoAP-EAP is able to complete the bootstrapping up to 6 hops. This is mainly due to it has a shorter message length and, therefore, it generates less fragmentation. Additionally CoAP has a less aggressive retransmission policy than PANA, which promotes sending less traffic over a constrained network.

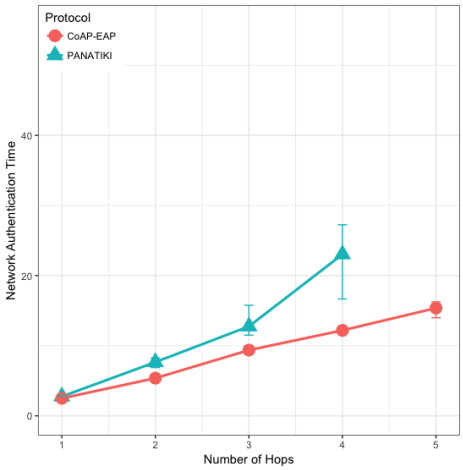
To corroborate this, Figure 3.8a–c show how the success percentage, the bootstrapping processes really finished from those that were started, evolves as the packet loss ratio increases. CoAP-EAP demonstrates a better performance in every packet loss ratio in comparison with PANATIKI. In the worst case scenario with 0.2 packet loss ratio, PANATIKI with 2 hops (1 intermediary node) already has a success of ≈9% which makes the bootstrapping service impractical, whilst CoAP-EAP in the same conditions is able to finish ≈85% of the initiated bootstrapping processes and keeps an acceptable success percentage as the number of hops increases.

Message Processing Time

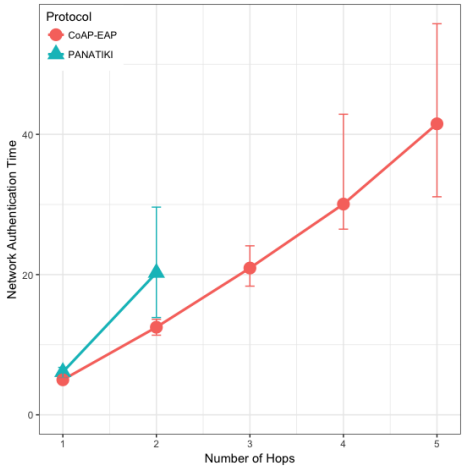
We have measured the processing time of the messages in the Smart Object (the most constrained device). This time includes the processing of a request and the time that the EAP state machine takes to process the EAP message. The response time measures the time it takes to create the response and send it to the Controller. The results are shown in Figure 3.9 for each message exchange.



(a) 0.0 loss ratio

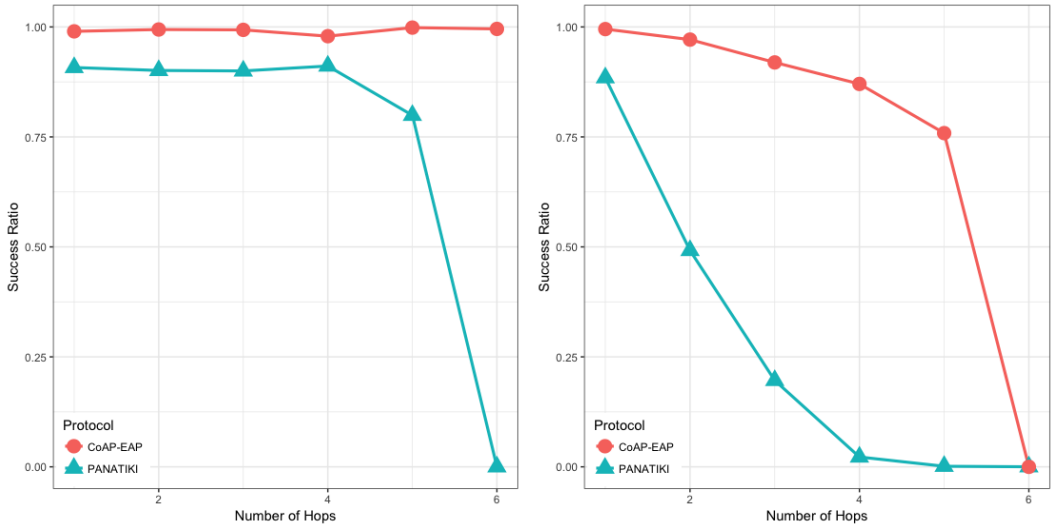


(b) 0.1 loss ratio



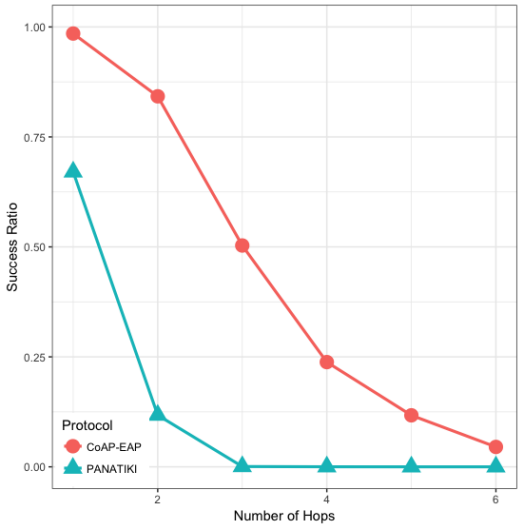
(c) 0.2 loss ratio

Fig. 3.7 Bootstrapping median time



(a) 0.0 loss ratio

(b) 0.1 loss ratio



(c) 0.2 loss ratio

Fig. 3.8 Success Ratio

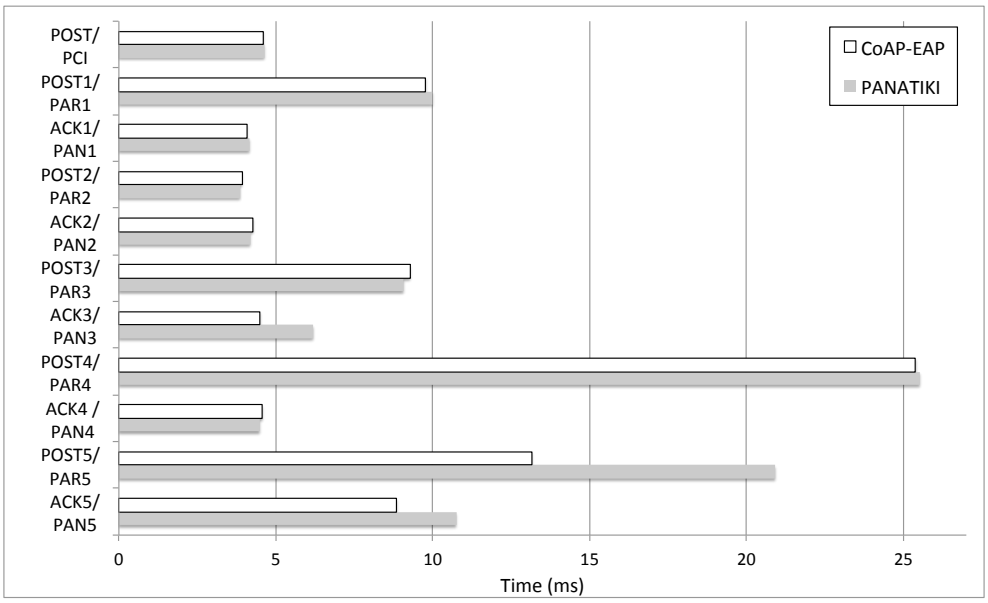


Fig. 3.9 Contiki message processing time.

A small advantage can be seen in the last exchange, where the AUTH check and key generation are done in both CoAP-EAP and PANATIKI. This is motivated by two reasons. First, the PANA message is longer than CoAP message and the cryptographic operations are performed over a longer message and, therefore, the cryptographic operations are performed over a longer message. Second, the processing time also includes the PANA_AUTH_KEY generation in PANATIKI and the CoAP _PSK generation in the case of CoAP-EAP. The CoAP _PSK generation is simpler since it involves fewer parameters. For example, PANA includes the first two messages (PAR/PAN) in the key derivation (Section 5.3 in [61]). The reason is that a cryptographic algorithm negotiation process is performed in these two messages and confirmed in the key derivation process. In CoAP-EAP, we do not include this, since our assumption is that a particular KDF will be used and selected by the controller, so no cryptographic negotiation is performed (see Section 3.3.6). In short, the CoAP _PSK derivation is simpler than PANA_AUTH_KEY.

With a total average processing time of ≈104 ms for PANATIKI and 93 ms for CoAP-EAP, a ≈11% of reduction in the processing time can be appreciated in CoAP-EAP over PANATIKI in total.

However, if we observe only the EAP lower-layer (the EAP-PSK processing time is common in both alternatives), with a total average processing time of ≈69 ms for PANATIKI and ≈59 ms for CoAP-EAP we can confirm an improvement of ≈15% in CoAP-EAP to process the messages. Although we can observe that CoAP-EAP takes less time to process the messages, this improvement is limited by the EAP method, which has an important

Table 3.3 Implementations memory size.

	Empty Main	Network Support (e.g., IP/UDP)	EAP	Lower Layer	Total Size
PANATIKI	62.7 kB	24.9 kB	9.4 kB	5.9 kB	102.9 kB *
CoAP-EAP	62.7 kB	24.9 kB	9.4 kB	3.8 kB (+4.6 kB cantcoap)	105.4 kB

* This size does not include any CoAP implementation.

weight in the message processing time. In other words, the EAP method implementation has a key impact because it limits the level of improvement. This leads us to conclude that it is important to design very lightweight EAP methods especially adapted for IoT networks.

In any case, if we contrast these values with the effect of fragmentation and message size in the bootstrapping time, we can see that this message processing time is practically negligible.

Memory Footprint

In terms of the memory footprint in the Smart Object, Table 3.3 shows the size in bytes of several components of the bootstrapping service based con CoAP-EAP or PANATIKI. To obtain these values, we have compiled with memory optimization (-Os compiler option) a set of programs that includes incrementally different modules (EAP, network support, EAP lower-layer) to estimate the size with each new module. Clearly, the EAP state machine and the corresponding modules for enabling network connectivity are also the same.

We observe that the total size of CoAP-EAP solution is slightly bigger than the PANATIKI one. The main reason is the inclusion of cantcoap library, which implements CoAP. Nevertheless we argue that CoAP implementation will surely be present in the majority of the Smart Objects as a common module to be used for other services in the Smart Object, not only for the bootstrapping service. Thus, this library, which adds 4.6 kB to the overall size, is likely to be reused. In fact, it is reasonable to think that PANATIKI will also need to include this library in real deployments to support other CoAP-based services (e.g., Ohba *et al.* [43] requires CoAP also for pulling cryptographic material) giving a total of 102.9 kB + 4.6 kB = 107.5 kB.

Let us illustrate this with an example. Let us assume we have a size of X kB available in the Smart Object. Part of that space is occupied by the operating system and the IP/UDP stack for communications. Of course, it is common in both alternatives (62.7 kB + 24.9 kB in our estimation in Table 3.3). The rest of the available space is to deploy services and applications in the Smart Object. For this, the Smart Object will ship a CoAP implementation (e.g., cantcoap) that will be used for the specific implementation of other services (e.g., a service to obtain temperature measurements). In our case, this CoAP implementation is cantcoap and

subtracts 4.6 kB from the free memory available. If we want now to support bootstrapping, we need additional source code. PANATIKI subtracts 5.9 kB but since CoAP-EAP is re-using CoAP implementation it only subtracts 3.8 kB. Thus, CoAP-EAP would save 2.1 kB with respect to PANATIKI in the available space for other services in a more realistic case.

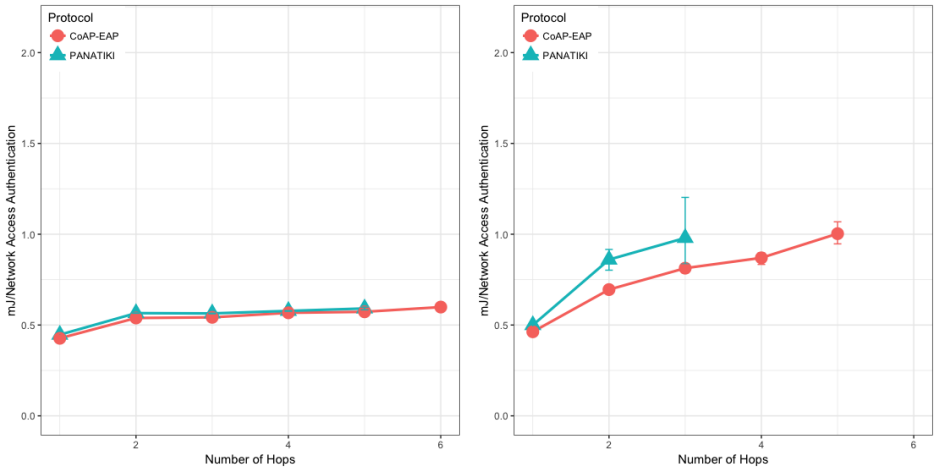
Energy Consumption

To obtain the energy consumption, we used the Powertrace [51] tool that comes with Cooja simulator. We used it to estimate the median energy consumed by each bootstrapping (mJ/bootstrapping) in CoAP-EAP and PANATIKI. The different measurements show the CPU consumption when the Smart Object is fully operative; the consumption when transmitting (TX) and receiving (RX) consumption and, finally the total energy consumption of each solution. Figures 3.12a–c, 3.11a–c, 3.12a–c and 3.11a–c show the energy consumption expressed in millijoules per bootstrapping (mJ/bootstrapping) in the same cases as Section 3.4.2.

As observed in all the measurements, as the packet loss ratio increases, more retransmissions (and, therefore, more message processing) are required (processing retransmitted messages, sending the retransmitted messages, *etc.*). Thus the energy spent because the CPU is working also increases. Additionally, we can see that the CPU consumption remains very similar in both alternatives when packet loss ratio is around 0. The reason is that the number of retransmissions is low and basically this CPU energy is spent on processing the messages shown in Figure 3.9. Since these values are very similar it is reasonable to observe similar CPU energy consumption. However, when the network conditions worsen, the CPU needs to work more in the case of PANATIKI than CoAP-EAP, which is an evidence that more retransmissions are required in PANATIKI due to more fragmentation as a consequence of the message length, and due to the more aggressive retransmission policy.

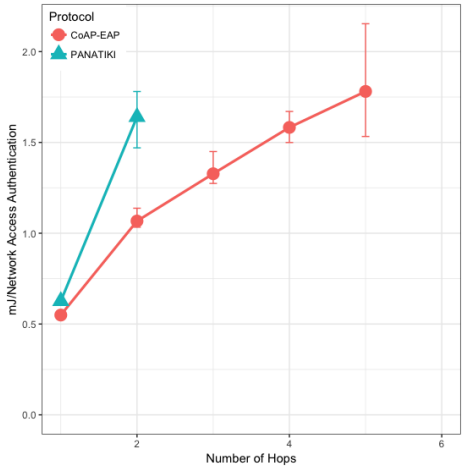
The energy spent transmitting (TX) (Figure 3.11a–c, as well as as receiving (RX) messages (Figure 3.12a–c over the radio interface, are the most energy consuming tasks. In fact, both are more important than the energy consumed by the CPU.

If the circumstances are not adverse, the best case scenario with 1 hop and 0% packet loss ratio, the CoAP-EAP behavior ($\approx$5.2 mJ/bootstrapping) is slightly better than PANATIKI ($\approx$6.3 mJ/bootstrapping). However, CoAP-EAP clearly reduces the energy consumption in contrast with PANATIKI when the network conditions are worse. For example, with 3 hops and 0.2 packet loss ratio PANATIKI spends around $\approx$60 mJ/bootstrapping in RX while CoAP-EAP spends $\approx$32 mJ/bootstrapping. This energy consumption is also evidence that PANATIKI involves more retransmissions and needs to send and, therefore, receive more messages.



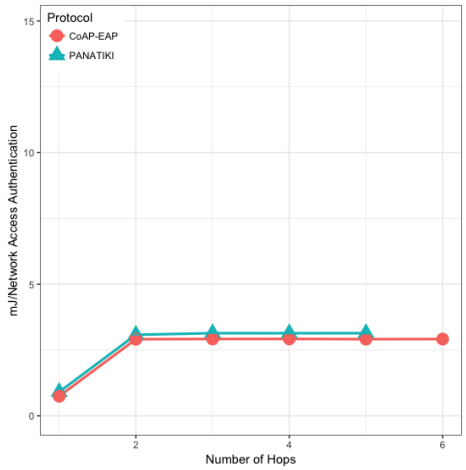
(a) 0.0 loss ratio

(b) 0.1 loss ratio

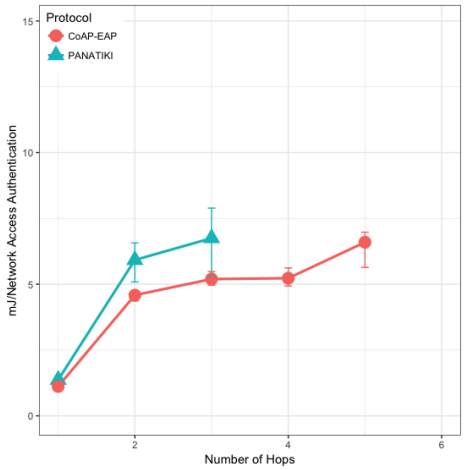


(c) 0.2 loss ratio

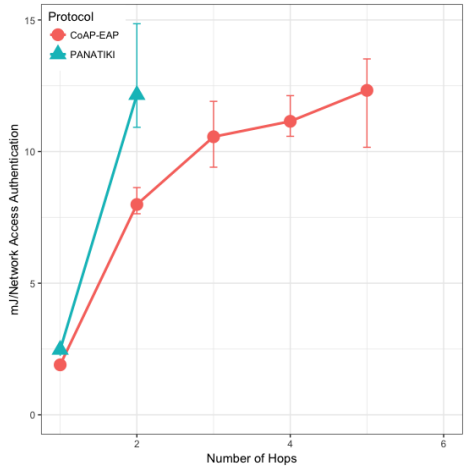
Fig. 3.10 CPU energy consumption



(a) 0.0 loss ratio

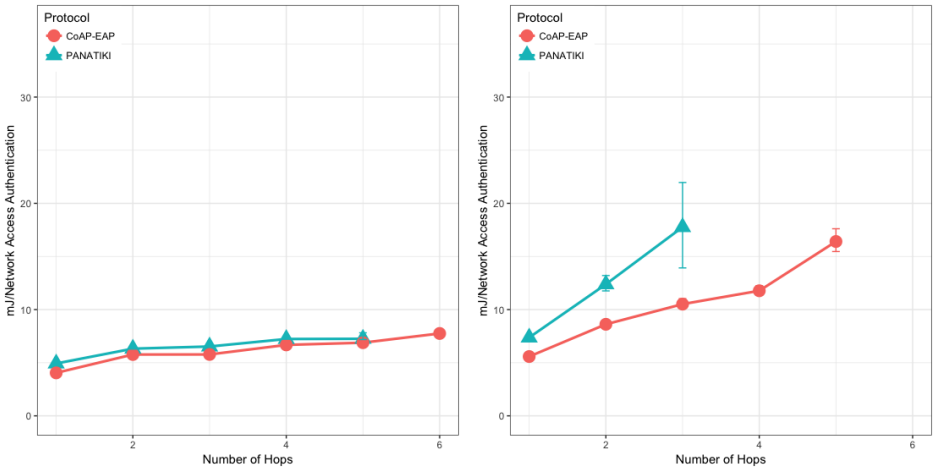


(b) 0.1 loss ratio



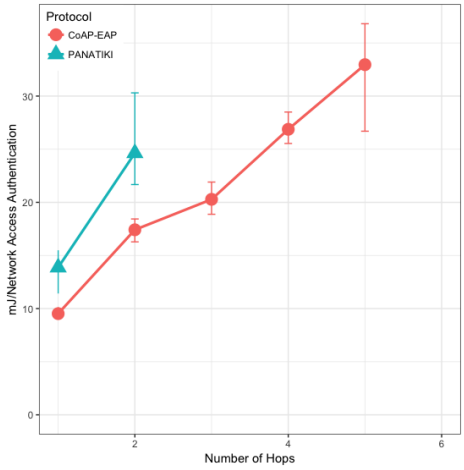
(c) 0.2 loss ratio

Fig. 3.11 TX energy consumption



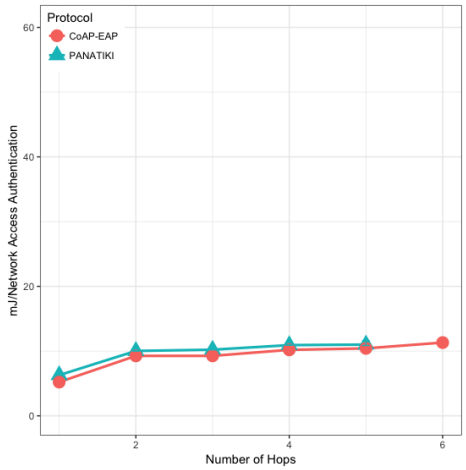
(a) 0.0 loss ratio

(b) 0.1 loss ratio

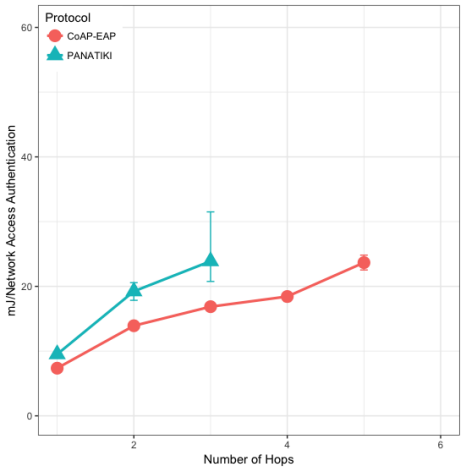


(c) 0.2 loss ratio

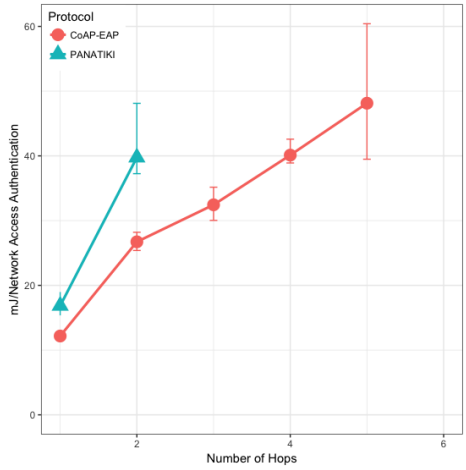
Fig. 3.12 RX energy consumption



(a) 0.0 loss ratio



(b) 0.1 loss ratio



(c) 0.2 loss ratio

Fig. 3.13 Total energy consumption

In the majority of scenarios with a loss ratio greater than 0, CoAP-EAP proves to be significantly better in terms of energy consumption in each of the modes.

Finally, Figure 3.13a–c show the total energy consumption per bootstrapping including all these factors, where we can observe that CoAP-EAP proves to be a more energy-saving solution than PANATIKI.

3.5 Conclusions

In this chapter we have presented a novel bootstrapping service, named CoAP-EAP, for large-scale IoT networks. The bootstrapping service is built on three main pillars. AAA infrastructures, EAP and CoAP. CoAP-EAP defines a new and simple EAP lower-layer using CoAP to provide a flexible, scalable, secure and constrained solution. This chapter presents the architecture and general flow of operation, and how, after the bootstrapping is completed, the smart object and Controller can establish a security association to secure their communications. Finally, this chapter shows a performance evaluation, considering the message length, bootstrapping time, message processing time, memory footprint and energy consumption. These results are contrasted with those obtained with PANATIKI, an optimized implementation for bootstrapping in the IoT based on PANA. We conclude that our solution, thanks to having a shorter message size brought to the overall bootstrapping process, provides substantial improvement in measurements such as bootstrapping time, the probability of finishing the bootstrapping (success percentage) and energy consumption. The results show a reduction of $\approx 50\%$ percent in the number of bytes used by the bootstrapping protocol, a reduction in bootstrapping time up to $\approx 56\%$ and a reduction in energy consumption up to 46%. Other improvements such as memory footprint and message processing time are more limited, mainly due to the EAP method, even when EAP-PSK is considered lightweight. This suggests that more constrained methods may be required in the future. Nevertheless, when we focus on one of the key contribution of our work, the proposed CoAP-based EAP lower-layer, we observe that a part is substantially reduced in compared to PANA.

In the next chapter we explore a new area of applicability for the bootstrapping service: Low Power Wide Area Networks (LPWAN), where reducing the message size is vital. There, we redesign and optimize the bootstrapping service to the constraints of these networks to provide a link-layer independent and lightweight bootstrapping service for LPWAN.

Chapter 4

A CoAP-Based bootstrapping service for LPWAN: LO-CoAP-EAP

This chapter introduces the second main contribution of this dissertation: *Low Overhead CoAP-EAP (LO-CoAP-EAP)*. The fairly recent set of technologies, known as LPWAN imposes very high restrictions on the link, beyond those observed in WPAN and for which we need to redesign and optimize CoAP-EAP to take into account these constraints and provide a bootstrapping service to these technologies. In this chapter, we review the state of the art in bootstrapping in LPWAN and show that, currently, each technology defines its own security mechanisms. Some of them provide a process to join the network and key management, others rely on pre-shared key material between the smart object and the network server and lacking interoperability. After that, LO-CoAP-EAP is described, designing its architecture and flow of operation between the Smart Object and the Controller. Then, we explain how specific security association protocols (SAP) in LPWAN can run using key material derived from the LO-CoAP-EAP bootstrapping. Finally, we show the performance evaluation and compare these results with CoAP-EAP.

4.1 Introduction

The wireless technologies that have dominated the IoT landscape have been mostly Bluetooth [72] or IEEE 802.15.4 [7] (also known as LR-WPAN) encompassed in what are known as Wireless Personal Area Networks (WPAN). In addition to the aforementioned wireless technologies, there has been a recent impulse toward LPWAN [176]. LPWAN encompasses wireless technologies for long-range communications. LPWAN allows small pieces of information to be sent between the IoT devices and an antenna up to several kilometers

away with small energy consumption, so expanding the impact of IoT applications on a larger scale. Among such applications, we can mention the connection of alarms to the cloud, smart farming and precision agriculture [176]. As a downside, LPWAN only supports data rates as low as 50 b/s or 250 kB/s, frame sizes as low as eight bytes and restrictions to access the medium with a duty cycle of 0.1% to 10% on some ISM bands [129]. In other words, apart from the constraints observed in LR-WPAN, LPWAN adds a very constrained communication link, beyond those observed in LR-WPAN.

Since the concept around LPWAN looks promising, some research groups and alliances (e.g., LoRaWAN [197], Sigfox [192], DASH [210], *Wireless Smart Ubiquitous Network* (WI-SUN) [104], etc.) are investing their resources in defining their own specific solutions to cover the downsides of LPWAN quickly. Very recently, the Internet Engineering Task Force (IETF) created a *IPv6 over Low Power Wide-Area Networks* (lpwan) Working Group to determine how groups can contribute to the evolution of LPWAN by defining technology independent protocols and solutions that favor interoperability between the different wireless technologies involved in LPWAN. For example, modifications to the 6LoWPAN standard are under consideration to adapt IPv6 to LPWAN [128]. Another example is the adaptation of the Constrained Application Protocol (CoAP) [190] to LPWAN. While there is an effort to homogenize communications in this type of network based on IPv6 and CoAP, little has been done to design a unifying solution that provides a common and well-defined model for bootstrapping for LPWAN networks, independent of the wireless technology. Nevertheless, this process involves authentication, authorization and key management which are also fundamental since may allow operators and network administrators to decide whether a particular device can or cannot join the network.

In this sense, the LPWAN WG has discussed the use of the AAA infrastructures to support bootstrapping [59, 65, 129, 102]. The reason is that LPWAN networks look very similar to current cellular deployments, but with very limited resources. In fact, current wireless networks such as 3G, *Worldwide Interoperability for Microwave Access* (WiMAX) or WiFi use, traditionally, AAA infrastructures to control the access to the network service on a large scale [16, 74, 58]. These infrastructures have been based on two main protocols: Remote Authentication Dial-In User Service (RADIUS) [167] or Diameter [57]. In conjunction with AAA, the Extensible Authentication Protocol (EAP) [10] is also considered to authenticate the devices because it provides a wireless technology independent protocol for flexible authentication and robust key management [12]. By definition, the EAP key management framework allows obtaining key material to derive new key material to bootstrap a security association in a specific wireless technology.

In this chapter, we propose a first solution for bootstrapping in LPWAN named Low-Power CoAP-EAP (LO-CoAP-EAP) based on the design defined in CoAP-EAP. The use of AAA infrastructures provides scalability and roaming for the bootstrapping; EAP allows different smart objects and organizations to use different authentication mechanisms (based on symmetric keys, certificates, etc.) according to their requirements; and CoAP, which is used as a constrained transport for EAP between the smart object and the network which are both connected through a link with a very limited bandwidth. The general design of LO-CoAP-EAP starts from our previous work in Chapter 3, named CoAP-EAP, but it is redesigned to take into account the constraints in LPWAN. In particular, LO-CoAP-EAP provides a CoAP-based service for bootstrapping in LPWAN. The new design of LO-CoAP-EAP reduces the number of exchanges and the overall number of bytes sent in the constrained link. As a collateral effect, we will also prove that LO-CoAP-EAP substantially improves CoAP-EAP in LR-WPAN networks.

The rest of this chapter is organized as follows. Section 4.2 details the related work. Section 4.3 details the proposed bootstrapping service, LO-CoAP-EAP: the architecture and operation. In Section 4.4, we show a performance evaluation including message length, authentication time, the proportion of successful authentications (success ratio) and the energy consumption in two test-beds: (1) a real deployment of a LPWAN network based on LoRa technology to get in-the-field measurements; (2) a network based on 6LoWPAN using the Cooja simulator to also show the impact in LR-WPAN networks. With these two test-beds, we have compared LO-CoAP-EAP against existing related work. Finally, we provide conclusions in Section 4.5.

4.2 Related Work

Bootstrapping for IoT is described in Heer *et al.* [81] and Garcia-Morchon et al. [69] as part of the process of a smart object joining an IoT network securely, at a specific time and place. This process entails the authentication and authorization of the smart object, as well as the transfer of security parameters (e.g., key material) for a trustworthy operation in the security domain.

In general, although most of the current work for bootstrapping in IoT has been devoted to LR-WPAN, some of these (specified below) deserve attention due to the relation to our solution. Large-scale scenarios use AAA infrastructures and EAP, while small to medium scale do not involve AAA infrastructures. For example, Garcia-Morchon *et al.* [69] in the case of the centralized scenario point out EAP as a candidate to perform the authentication and generation of key material, proposing the Protocol for carrying Authentication for

Network Access (PANA) [61] as a standard and link-layer independent EAP lower layer. In fact, the ZigBee IP standard [217] uses PANA as a protocol for bootstrapping. As such, our proposal LO-CoAP-EAP can be compared with PANA, as we will analyze in Section 4.4. O'Flynn *et al.* [144] consider also a centralized architecture using EAP and PANA or 802.1X [1] as EAP lower layers, while Sarikaya [180] and Sarikaya et al. [178] propose EAP-TLS concretely as the authentication protocol, but without interaction with an AAA infrastructure. S. Das et al. [42] also propose a centralized alternative using PANA, EAP and AAA to bootstrap a PSK to establish a Datagram Transport Layer Security (DTLS) [162] or Internet Key Exchange (IKEv2) [106] security association between the smart object and the PANA Agent (PAA), using afterwards CoAP for the normal operation. Alternatively, Moreno *et al.* [175] designed and implemented a lightweight version of a PANA client (PaC) for Contiki O.S. [52] (PANATIKI) by adapting PANA for constrained devices. Finally, Garcia-Carrillo and Marin-Lopez [66] proposes a technology independent EAP lower layer using CoAP to transport EAP messages (CoAP-EAP) that improves the PANA-based solutions, to leverage the features of CoAP as a suitable protocol for communication between constrained devices and use of AAA infrastructures to provide scalability and federation capabilities.

There are also solutions that use EAP adapted to IoT, by either modifying EAP itself, or defining new EAP methods more optimized for the constraints of IoT networks. However, these solutions imply that wireless technologies where EAP is applied need to define an EAP lower layer specific to the technology. For example, it would imply that existing LPWAN technologies should design their own link-layer EAP lower layers to transport those EAP methods, which is far from the general case. On the contrary, LO-CoAP-EAP offers a wireless independent EAP lower layer to avoid this problem and allows existing or new EAP methods for IoT to be used without modifying the underlying technology.

Thus, the constraint LPWAN imposes on the network link requires adapting or redesigning existing solutions. Most of the LPWAN technologies are fairly recent and are still under development, and as a consequence, solutions related to security and bootstrapping are still immature. Specifically, some of the alliances and organizations working in the area of LPWAN propose solutions to authenticate and secure the communications using pre-shared keys. Additionally, the key management is usually handled in a static manner, with no dynamic key distribution, and there is little information about how the necessary keys will be managed, installed, refreshed, etc., or how the devices will be authenticated to access the network and how they will interact securely with the infrastructure. For example, DASH [210] includes the notion of authentication and access control, defining three users (root, user and guest) and authentication keys root and user, as well as a network key. There is no specific mention about the method used to generate the aforementioned keys, if they are configured

Table 4.1 Crypto suites used by LPWAN technologies to authenticate and encrypt the frames.

Technology	Authentication/Integrity Algorithm	Payload Encryption Algorithm	Key Management Protocol
LoRaWAN	AES-128 CMAC	AES-128	YES
Sigfox	N/A	N/A	NO
IEEE802.15.4k	N/A	N/A	-
IEEE802.15.4g	AES-CCM	AES-CCM	-
Random Phase Multiple Access (RPMA)	Twoway 16-byte hash	256-bit	-
DASH-7	CBC-MAC-128/64/32 AES-CCM-128/64/32	AES-CTR	-
Weightless	AES-128/256	AES-128/256	YES
NB-LTE	AKA-128	AKA-128	YES

manually, or provided dynamically using other methods of key management. Sigfox [192] provides integrity of the communications, with encryption at the application level, and it considers the future integration of a secure element to store key material. However, there is no mention of how these keys are installed, either manually or through dynamic key management methods. IEEE 802.11ah [110] defines a modification in the scheduling of the bootstrapping over the IEEE 802.11ai amendment, but no references to the modifications of the bootstrapping methods are made, so we consider that they are the same as defined for IEEE 802.11. LoRaWAN [197] does include a join procedure to authenticate the IoT devices based on pre-shared key Application Session Key (AppSKey) to derive fresh keys (NetworkSKey and AppSKey) to protect the LoRa link between the IoT device and the network. Recent work ([65] and Diameter [102]) has defined a way to integrate the LoRaWAN join procedure with AAA infrastructures to provide scalability and roaming support. However, these solutions are tailored to LoRaWAN.

Table 4.1 summarizes the current state of several technologies related to LPWAN security and bootstrapping, showing the algorithms used to provide integrity and encrypt the messages and if the technology provides a key management protocol to derive fresh key material to protect the link. There is a common occurrence of the use of symmetric cryptography, understandable due to its properties, providing security at a computationally low cost in comparison with asymmetric cryptography, and furthermore, the existence of hardware implementations of the most common cryptosuite, AES, increases its efficiency.

Thus, to the best of our knowledge, this is first work to contribute a solution for bootstrapping for LPWAN that tries to deal with the process in a homogeneous way through different technologies.

4.3 Bootstrapping in LPWAN: LO-CoAP-EAP

4.3.1 Requirements of the Service

The general requirement that leads to the design of the LO-CoAP-EAP service is to have a bootstrapping service for LPWAN regardless of the underlying radio technology being used, relaying on current standards, with the following characteristics:

- Flexible authentication mechanism: The service has to provide the needed flexibility for the authentication due to the variety of operators, technologies and requirements of each deployment.
- Operational homogeneity: The service had to be built on top of a protocol that is common to most of the IoT devices.
- Link-layer independent solution: The solution should be independent of the link-layer technology to be supported by the set of LPWAN technologies.
- Reduce overhead: Due to the high restrictions of LPWAN in terms of bandwidth, we need a solution with a reduced overhead, saving as much messages and bytes sent over the link as possible.
- Bootstrapping subsequent security association protocols: As each LPWAN technology may rely on different protocols to secure its data communications, we need to provide key material for different protocols to secure subsequent communications between the smart object and the network.

4.3.2 The LO-CoAP-EAP Service

Low-Overhead CoAP-EAP (LO-CoAP-EAP) is a CoAP-based and link-layer independent bootstrapping service designed considering the constraints imposed by LPWAN networks. LO-CoAP-EAP builds on top of three main technologies: CoAP, AAA and EAP. We use CoAP as a lightweight application protocol for constrained devices, allowing reusing source code in existing smart objects since CoAP is a common piece in IoT stacks for Machine-to-Machine (M2M) communications [147]. The integration with AAA infrastructures provides

scalability, roaming/federation support and a common CoRE independent of the technology to centralize the authentication and key distribution procedures. With EAP, we have flexibility to choose the authentication method, depending on the requirements of different organizations. Additionally, the EAP Key Management Framework (EAP KMF) provides the rules for key derivation to bootstrap security associations for different technologies, to protect data traffic in the constrained link.

Next, we describe the architecture of LO-CoAP-EAP network authentication service and its operation, and we detail the main changes with respect to our previous work in order to adapt the solution to cover LPWAN networks.

4.3.3 Architecture

LO-CoAP-EAP defines three main entities in its general architecture: the smart object, the controller and the AAA server. The smart object is the target of the authentication, and the controller manages the network. It offers services to the security domain and intermediates during bootstrapping between the smart object and the AAA server. Finally, the AAA server is in charge of the authentication and granting permissions for the requested services. LO-CoAP-EAP entities integrate EAP, CoAP and AAA as follows: the smart object instantiates an EAP peer and a CoAP client and server; the controller integrates an EAP authenticator, a CoAP server and client and an AAA client to interact with the AAA server (authentication server) of the identity provider. It is worth noting that there can be one or more AAA servers between the controller and the authentication server.

The IETF lpwan WG defines in its overview document [59] a generic terminology for the architecture. They define end-devices as the things, sensors, actuators, etc. The radio gateway is the end-point of the constrained link. The network gateway (or network server) is the interconnection between the radio gateway and the Internet. Finally, LPWAN-AAA is the entity in charge of managing the authentication. The LO-CoAP-EAP architecture maps to their LPWAN counterparts as follows: the smart object is the end-device; the controller acts as the network server; and the AAA server as the LPWAN-AAA.

4.3.4 General Operation

To refer to the CoAP-based service for network authentication, LO-CoAP-EAP uses the URI coap://<controller-IP>/b. The smart object, acting as the CoAP client, initiates the LO-CoAP-EAP exchange. Without loss of generality, we use RADIUS as the AAA protocol in the description. The operation, shown in Figure 4.1, is as follows:

The smart object sends a POST message (Step 1) with the no-response option [26]. This option allows the smart object to signal it does not expect a response to this request. In this first message, the smart object sends a random number nonce-s carried in an option named nonce and the smart object's identity in *Network Access Identifier* (NAI) format [46] (user@domain.org) in the payload. After this first message, all the remaining exchanges will be done with the controller acting as the CoAP client and the smart object as the CoAP server. This role choice follows the guidelines in [115] to simplify the implementation assuming that the smart object will be more constrained than the controller. The controller can now process the smart object' identity and send it to the AAA server in a RADIUS access-request (Step 2). Typically, the identity is transported in a EAP response/identity, but this implies sending an EAP request/identity first. To save this exchange, the RADIUS (and Diameter) standard allows the smart object's identity to be carried into an attribute called user-name instead of EAP-response/identity message. In this case, the attribute EAP-message, which contains EAP messages, will be empty in this message [11, 53]. Additionally, it will include a NAS-port-type. This attribute indicates the physical port of the controller that is authenticating the user, and it is useful for the AAA server to know whether the access is being performed over an LPWAN link, which may modulate the authorization decision and the delivery of configuration parameters for the controller. After the AAA server processes this message, it decides what EAP method is to be used based on the smart object' identity; let us call it EAP-X. Then, it responds with the first message of the EAP method (EAP-X 1) embedded in the attribute EAP-message of a RADIUS access challenge (Step 3). When this message arrives at the controller (Step 4), it decapsulates EAP-X 1, encapsulating it in the payload of a confirmable POST message. The token value in CoAP is set to empty to further reduce the length of the message. The smart object answers this POST with a piggybacked response that contains the EAP response (EAP-X 2) (Step 5). The controller will then forward EAP-X 2 to the AAA server using a RADIUS access-challenge (Step 6), and so on. This process continues until the EAP method finishes (Steps 7–10), the number of exchanges depending on the EAP method. Finally, the AAA server sends a RADIUS access-accept with an EAP success. Along with the EAP success, the MSK is delivered to the controller with a network access lifetime (i.e., session-timeout) (Step 11). Then, the controller sends a confirmable POST message that contains a nonce nonce-c in the nonce option and authorization data like the session lifetime (Step 12) in the payload.

Typically, the EAP success is also included in this message. However, we do not include it in order to save bandwidth, since the EAP standard allows the EAP peer (smart object) to proceed without receiving the EAP success, but with an alternate indication of success [10]. This indication happens in two ways: (1) the reception of the confirmation POST message

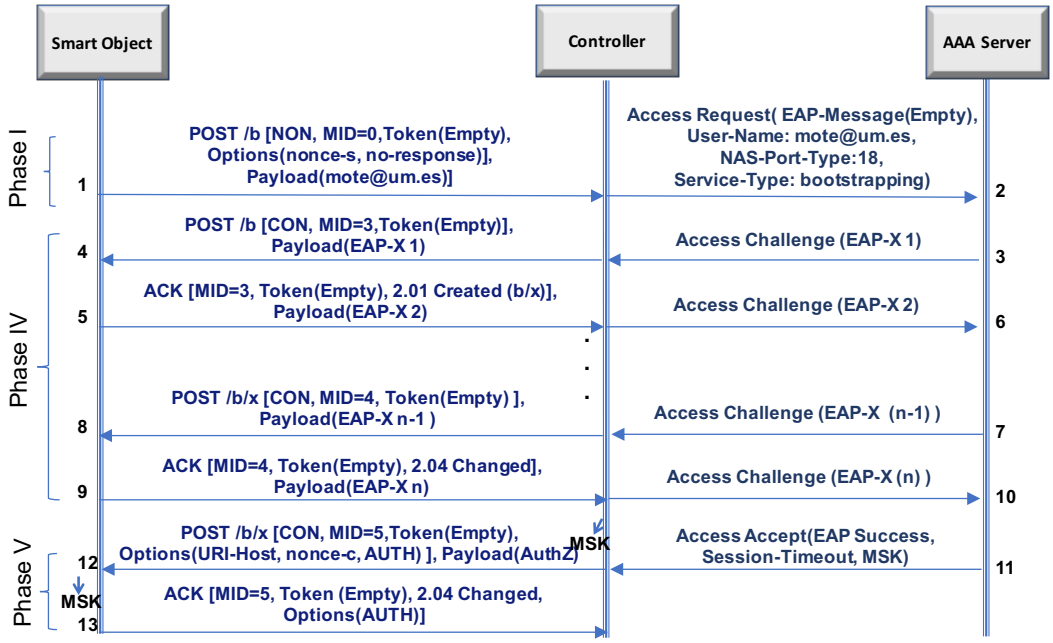


Fig. 4.1 LO-CoAP-EAP bootstrapping service flow with generic EAP method (EAP-X)

without EAP and with the AUTH option is an indication that the controller considers the EAP authentication finished. Second, the smart object knows that the EAP authentication went well if an MSK is available. Nevertheless, both entities still need to prove the possession of the MSK as mentioned in the EAP Key Management Framework (EAP KMF). It is worth noting that this last exchange (12–13) is integrity protected by an authentication tag embedded in an AUTH option. This provides a simple way of providing proof-of-possession of the MSK between the smart object and the controller.

To protect the communications between the smart object and the controller, after the LO-CoAP-EAP authentication service is completed, our solution is open to run the Security Association Protocol (SAP) for any particular LPWAN technology, providing fresh, shared key material with the key derivation capabilities of the EAP KMF, as we elaborate in the next section.

The reader will note that there is a division of the exchanges in different phases in Figure 4.1. This division is there to compare easily the changes done over CoAP-EAP elaborated in Section 4.3.6, where we describe each phase and its function.

4.3.5 Bootstrapping Security Associations for LPWAN

In this section, we specify how to derive key material from the MSK to secure the communications between the controller and the smart object. The derived keys are known as

Transient Session Keys (TSKs) [12] in EAP lingo. Based on these keys, we can run virtually any security association that relies on pre-shared keys (in fact, existing wireless technologies such as WiFi or WiMAX already derive TSK s from the MSK). To illustrate this, we provide a simple example of this procedure based on LoRaWAN [197]. In particular, LoRaWAN requires the AppKey to run its security association protocol that involves two messages: join request/join response. Once the EAP authentication is successful, both the smart object and the controller share the MSK. From the MSK, we derive a TSK, which will be the AppKey in the LoRaWAN specification. For the derivation process, we use a similar KDF as the one specified in [173]. Specifically, we use AES CMAC-PRF-128 as the *Pseudo Random Function* (PRF), which uses AES-CMAC-128 as a primitive. Both primitives use AES-128 as the building block since it is widely used in constrained devices. Then, as the KDF we use the function *PRF+* defined in [106], as recommended in [173]. The *PRF+* is able to generate key material of different lengths. The example of the derivation of the AppKey can be seen in Equation (4.1). The AppKey is a 16-byte length key. The input for the KDF (PRF+) is the following: the MSK derived from the EAP method; an ASCII code representation of the non-NULL terminated string "IETF_ LoRaWAN " (excluding the quotes), to which we concatenate the NULL value and the nonces exchanged. Sixty four is the length of the MSK; *length* is the length of the output (16 in the case of the AppKey).

$$AppKey = KDF(MSK, \text{“}IETF\_LoRaWAN'' \mid NULL \mid nonce-c \mid nonce-s, 64, length) \tag{4.1}$$

The same KDF can be used to generate key material for any other technology that requires key material to protect data frames. The unique changes would be the replacement of the label "IETF_ LoRaWAN " for another more appropriate one (e.g., IETF_SigFox) and the expected length of the key material.

As depicted in Figure 4.2, once both entities have derived the AppKey, they can simply run the LoRaWAN *Over The Air* (OTA) security association protocol that allows one to derive two keys, Application Session key (AppSKey) and Network Session Key (NwkSKey), to protect communications (see more details in [197]).

4.3.6 Main Changes to CoAP-EAP

For the design of LO-CoAP-EAP, we perform a set of changes to CoAP-EAP to achieve the primary target of this chapter: reduce the number of bytes dedicated to the bootstrapping

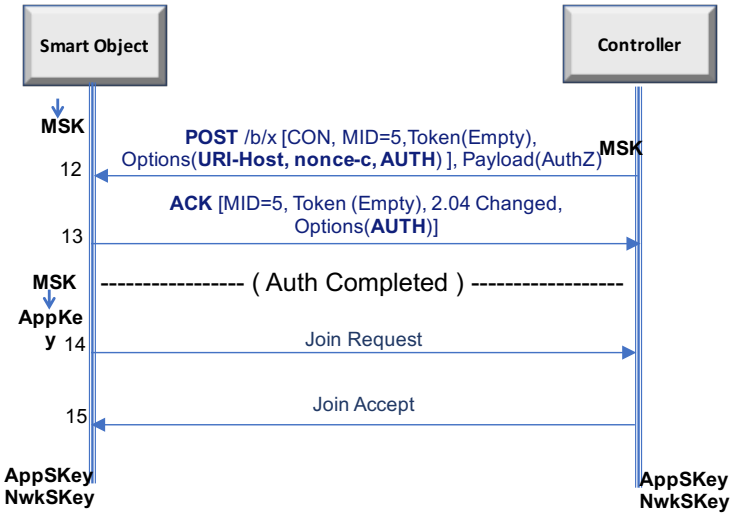


Fig. 4.2 LoRaWAN security association establishment after LO-CoAP-EAP authentication.

process traveling over the constrained link. The result is a reduction of the number of messages on flight and the size of the messages exchanged between the smart object and the controller.

To achieve this, we have analyzed each phase of CoAP-EAP in order to determine which parts can be simplified. In this sense, CoAP-EAP can be divided into five phases: (I) trigger; (II) exchange of nonces; (III) exchange of identity; (IV) exchange of EAP method; (V) sending the EAP success. Next, we see how LO-CoAP-EAP deals with each phase.

Figure 4.3 illustrates the changes in CoAP-EAP. The red (not highlighted) arrows signify that those messages are deleted in LO-CoAP-EAP, either because of being simplified or that information moved to other messages. The content in the messages is crossed out and colored in red, signifying that the content has been simplified. Common simplifications to all phases are: (1) the URI to identify the authentication service is reduced from /boot to /b to save three bytes in each request; (2) the token previously generated randomly, fixed for the duration of the exchange and used as session identifier is now set to empty. Discussion about the implications in LO-CoAP-EAP is shown in Section 4.3.7.

- Trigger (Phase I): To start the process in LO-CoAP-EAP, we send in the first POST used to trigger the authentication process the identity of the user in the payload of the message and the nonce-s in a new option (nonce option) for the controller. Additionally, this message carries a no-response option [26], which indicates to the controller that there is no need for a response of any kind to this request.

- Exchange of the nonces (Phase II): The nonce exchange that was previously done following the trigger message is avoided, and the nonces are embedded in other

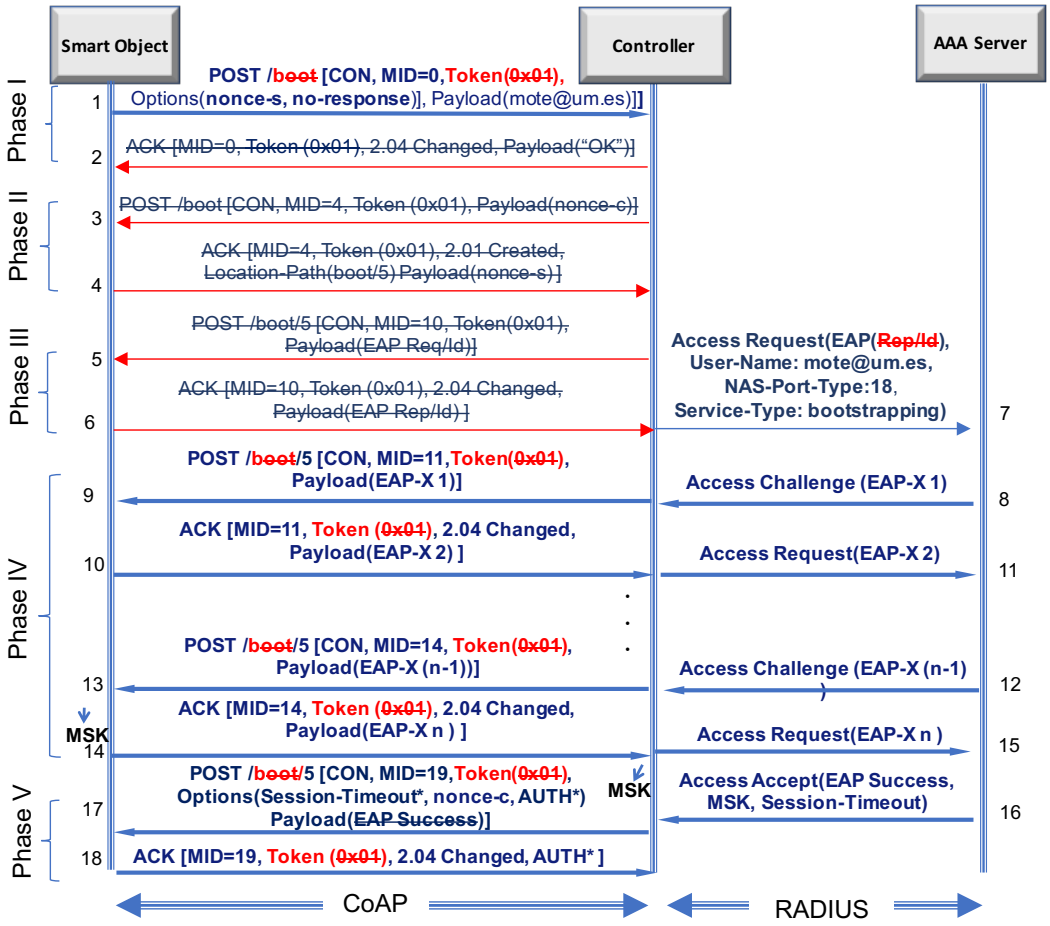


Fig. 4.3 Changes to the original CoAP-EAP.

messages. Although this saves a complete exchange, its original purpose is still considered to alleviate Denial of Service (DoS) attacks, as is discussed further in Section 4.3.7.

- Exchange of the Identity (Phase III): After the nonce exchange, the EAP identity was requested. This exchange is now avoided, since the EAP standard specifies this exchange as optional. The identity of the user, as mentioned previously, is now sent in the trigger message. With this change, we save another exchange.
- EAP method exchange (Phase IV): The messages involved in the exchange of EAP method exchange are not further altered beyond the simplifications mentioned before and run as expected.
- Sending the EAP success (Phase V): Sending the EAP success message to the smart object is also optional. Avoiding sending the EAP success in the last exchange, we save a four bytes. We only send the nonce-c to the smart object for key derivation purposes in that last request.

It is worth noting that, as in any type of authentication process, we need radio communication between the smart object and the controller for the LO-CoAP-EAP service to work. Additionally, communication between the controller and the AAA infrastructure is also needed to complete the process. This communication is done through the *Wide Area Network* (WAN) connection, not subject to the bandwidth limitations. Certainly, the EAP exchanges in the authentication involves an indirect communication between the smart object and the AAA server using the controller as the intermediary, but this process will only be done once; therefore, the smart object does not need constant connection with the AAA infrastructure after the authentication.

4.3.7 Additional Discussion

Session Identifier and Empty Token

Changing the token value to empty saves some bytes sent over the link related to the authentication service. However, we argue that this change does not affect the operation of the protocol.

Indeed, the purpose of the CoAP token is to correlate a CoAP request and response [190]. More specifically, it is intended as a client-local identifier to differentiate between concurrent requests. Based on this, we state how the LO-CoAP-EAP protocol still maintains its functionality in spite of setting its value to empty. Firstly, since EAP is a lock-step protocol (see Chapter 2), the LO-CoAP-EAP protocol that transports EAP is also designed

as lock-step. The reason is simple: to obtain network access through a specific controller, a single authentication process is enough. After all, the link is very constrained, and any traffic should be reduced. Therefore, the controller (CoAP client) will not send a new request until having received the corresponding response. Secondly, the original use of the token (as the session identifier) can be fulfilled by other means: we only need a known value by both parties that uniquely identifies the smart object. This can be done using the IPv6 address or the MAC address in case we consider a direct link between the smart object and the controller.

Analyzing Retransmissions

Retransmissions are a tool to provide reliability for the communications, sending again a message if there is no indication that the message arrived successfully. The retransmission policy suggested in the CoAP standard has been adapted for LR-WPAN networks, though some improvements may be still performed in the default policy [190]. In particular, the default policy specifies to wait approximately 2 s between when a message is sent and the ACKnowledgment arrives and increasing exponentially the retransmission time to double the previous retransmission time, up to a maximum of four retransmissions.

Nonetheless, this retransmission policy needs further considerations in radio technologies with a very low bit rate, such as LPWAN networks. For example, according to LoRa technology, assuming a Spreading Factor (SP) of 12, the bandwidth set to 125 kHz and the payload to the maximum allowed in LoRa (256 bytes) and setting the other variables as default in the Semtech LoRa Modem Calculator tool [187], a message will spend on the air approximately 7.5 s. As we can observe, before a CoAP message arrives at the other endpoint, a retransmission may be triggered (in some cases, more than one). Thus, it is necessary to adjust the retransmission policy in LPWAN networks.

We need to to assign the minimum elapsed time before a retransmission is sent. In CoAP, this is called ACK_TIMEOUT. Therefore, we need to consider the parameters used in LoRa when transmitting to get this number for the CoAP policy to establish its value. Basically, we need to know the Round Trip Time (RTT) of a message (the time it takes to arrive from a source to a destination and back) to establish ACK_TIMEOUT. Following the previous example and the implementation of the CoAP retransmission mechanism, we would set the ACK_TIMEOUT to 8 s (rounding up to cover processing time), which will give us a MAX_TRANSMIT_SPAN of 152 s, which corresponds with the time a client considers that a confirmable message was not received. Considering the results of the previous example, the controller will understand that a LO-CoAP-EAP authentication is discontinued after waiting 152 s for a piggybacked response.

LoRa technology has adaptive capabilities that need to be taken into account when establishing a retransmission policy for CoAP. As a side note, we comment that EAP has a retransmission mechanism that is disabled, since it is running over a our reliable lower layer LO-CoAP-EAP. Additionally, having a lower bit rate in the constrained link has an effect on the time each party involved in LO-CoAP-EAP needs to store any state related to it. The smart object, and its EAP state machine, needs to configure the amount of time to wait for a valid request before aborting (known as ClientTimeout in [205]). It is also the case for the AAA server that needs to keep the status related to the EAP authentication. If it is not stored for sufficient time, the AAA server might assume the authentication has failed, erasing the associated state, when it is simply just taking longer.

DoS Attacks

The new design of LO-CoAP-EAP keeps the same security properties of CoAP-EAP discussed in Section 3.6 in [66], except for one particular aspect: LO-CoAP-EAP may save two messages after exchanging the smart object's identity and use the smart object's identity in the very first message (Step 1 in Figure 4.1), so that the controller can contact the AAA server immediately.

Although the modifications in the first message save bits in the link, this creates an additional state (authentication state) in the controller beyond just storing the smart object's identity. In particular, this authentication state includes the EAP state machine, which must be initialized with different parameters [205], and the AAA client session required to communicate with the AAA server. This has an important implication: an attacker may blindly send the "trigger" message with different MAC or IPv6 addresses in a loop, therefore creating the authentication state for each trigger message sent to the controller.

When a link has unrestricted bandwidth, the number of messages starting authentications arriving at the controller may be very high. Each one will create some authentication state that the controller has to store for a determined time. This would provoke growth in the number of authentication states in the controller that surpass the capacity of the controller. However, if the capacity of the link is very reduced, then the maximum number of messages starting an authentication is dramatically reduced, as in the case of LPWAN [15], limiting the authentication state generated in the controller. For example, if we use LoRaWAN, when the devices behave respecting the duty cycle, we can expect less than one authentication request per second. In the worst case, if an attacker were to use the channel at full capacity, we can expect around sixteen messages per second, in each channel. A controller that has to manage thousands of legitimate devices is assumed to be able to manage this amount of states created by attackers. In any case, if the controller evaluates that it is creating states at an

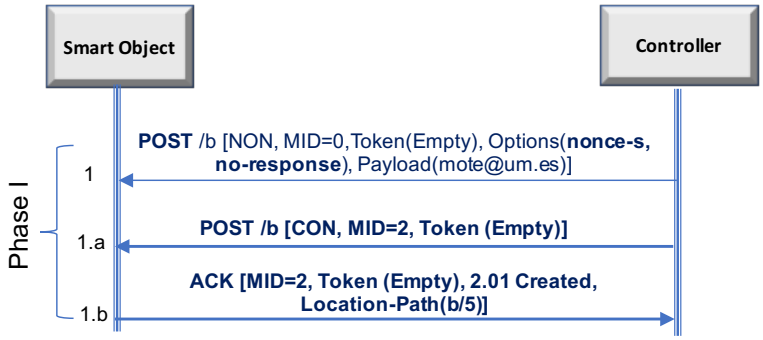


Fig. 4.4 LO-CoAP-EAP handshake.

abnormal rate, it can always perform the handshake to mitigate this effect. In summary, the link in LPWAN is the main bottleneck and most restricted resource, which limits the number of authentications per second that can start in the controller. In fact, sending additional messages may create a problem in the link regardless of the use of LO-CoAP-EAP.

Either way, in order to alleviate potential *Denial of Service* (DoS) attacks, the controller can always engage in an optional (it was mandatory in CoAP-EAP) handshake (1a and 1b in Figure 4.4) with the smart object before creating the authentication state (EAP and AAA). In this manner, the attacker cannot provoke the creation of the authentication state using a loop since it must answer correctly the message sent by the controller. In any case, it is up to the controller's policy to select when this handshake should be performed or not (which is out of scope of this work). For example, the controller may detect some irregular activity during the access (e.g., many triggers in a very short period of time) and, as a consequence, activate this handshake to avoid consuming resources for the authentication state.

4.4 Experimental Results

To test different technologies and conditions, we have performed evaluations in the Cooja simulator (Test-Bed 1) [146] and with real devices for LPWAN (Test-Bed 2). The first testbed is used to achieve three goals: (1) to get an approximation of the performance of the protocol from one to several IP hops with different loss ratios, providing hard conditions in the link, since there is ongoing work in LPWAN exploring the multi-hop case [22, 13, 21]; (2) to compare LO-CoAP-EAP with PANATIKI [175], which is a PANA implementation (a current standard for a link-layer independent bootstrapping in IoT) adapted to the Contiki O.S.; and (3) to give proof that LO-CoAP-EAP also provides important improvement with respect to CoAP-EAP in LR-WPAN. Based on the results of this first approximation, we have prepared a real LPWAN deployment using LoRa radio technology (a LoRaFabian [151]

Table 4.2 Comparison of the message lengths. PANATIKI, PANA implementation of Contiki.

		PANATIKI		CoAP-EAP		LO-CoAP-EAP with Handshake		LO-CoAP-EAP without Handshake	
Phase	**Message (CoAP-EAP)**	**LL**	**LL + EAP**	**LL**	**LL + EAP**	**LL**	**LL + EAP**	**LL**	**LL + EAP**
I	1)POST	16	16	13	13	29	29	29	29
II	2)POST(nonce-c)	40	40	18	18	6	6	-	-
	3)ACK(nonce-s)	40	40	20	20	8	8	-	-
III	4)POST(Request/Id)	43	48	16	21	-	-	-	-
	5)ACK(Reponse/Id)	41	60	9	28	-	-	-	-
IV	6)POST(EAP-PSK 1)	27	56	16	45	9	38	7	36
	7)ACK(EAP-PSK 2)	24	84	9	69	5	65	9	69
	8)POST(EAP-PSK 3)	25	84	16	75	9	68	9	68
	9)ACK(EAP-PSK 4)	25	68	9	52	5	48	5	48
V	10)POST(EAP success)	84	88	35	39	34	38	34	38
	11)ACK	52	52	27	27	23	23	23	23
% Reduction over PANATIKI		-	-	≈55%	≈36%	≈69%	≈49%	≈72%	≈51%
% Reduction over CoAP-EAP		-	-	-	-	≈32%	≈20%	≈38%	≈23%
Total		417	636	188	407	128	323	116	311

LL: Lower Layer message length.
LL + EAP : Lower Layer message length including EAP message length.

network), as a representative example of LPWAN, with real devices for LPWAN. Without loss of generality, we have obtained our results using a light and standard EAP method, EAP-PSK. This method consists only of four messages to complete the authentication. Since our solution is independent of the EAP method, any other method could have been used.

The parameters to be measured are: (1) message length, (2) network authentication time and success ratio (that is, the relation between finished and initiated authentications), (3) energy consumption and (4) memory footprint.

4.4.1 Message Length

In general, the message length is relevant in terms of the time the smart object takes to process it (including the time to send and receive messages over the network). In LPWAN, this is *Encapsulating Security Payload* (ESP) ecially relevant, taking into account the restrictions of LPWAN in the link.

Table 4.2 shows the comparison in terms of message length (in bytes) for each alternative: (1) PANATIKI, (2) CoAP-EAP, (3) LO-CoAP-EAP with handshake and (4) LO-CoAP-EAP without handshake. We detail the length of the EAP lower layer without the EAP message (LL) and including it (LL + EAP). PANATIKI is shown for reference since a detailed description of the message length comparison with CoAP-EAP is done in [66]. This will give a better understanding of the impact the redesign has on the reduction of the size of the protocol and, overall, the percentage of bytes saved in each case.

Overall, an important reduction in size of the lower layer can be appreciated. With CoAP-EAP over PANATIKI, we reduce up to ≈50% comparing the lower layer and up to ≈32% the lower layer + EAP messages. With LO-CoAP-EAP over PANATIKI, we reduce up to ≈70% compared with the lower layer,and up to ≈50% the lower layer + EAP messages. Comparing LO-CoAP-EAP with CoAP-EAP and with LO-CoAP-EAP with handshake, we have a reduction of ≈ 30% for the lower layer and a reduction of ≈22% over the whole protocol exchange (lower layer + EAP messages). For LO-CoAP-EAP with handshake, this reduction is ≈39% for the lower layer and ≈25% for the whole protocol exchange compared with CoAP-EAP. This reduction is important to consider in very constrained links such as LPWAN networks, as we show in Section 4.4.3.

4.4.2 LO-CoAP-EAP Performance in the Cooja Simulator

Figure 4.5 shows the test-bed we use in the Cooja Network Simulator with Contiki OS Version 2.7 [146]. The smart objects used are the Zolertia Z1 with 92 kB of nominal ROM when compiled with 20-bit architecture support and 8 kB of RAM. The compiler is msp430-gcc Version 4.7.2. The specifications of the computer used for running the test-bed are shown in Table 4.3. In terms of the software packages, we have used cantcoap [127] ported to C language since it gives us the flexibility needed to only create CoAP messages without including the REST engine integration, which is sufficient for our proof-of-concept implementation. In this test-bed, we use the *IPv6 Routing Protocol for LLN* (RPL) border router, which enables communication between the simulated Cooja Network and the outside physical network where the controller is located. Between the border router and the smart object, there can be zero or more smart objects. Having several hops between the smart object and the controller allow us to observe the behavior of the bootstrapping process in multi-hop networks and how each parameter is affected (network authentication time, success ratio and energy consumption), when intermediate smart objects act as IP-forwarders. Following the recommendations in [3], we have performed the simulations in Cooja with a randomly generated seed to automate running the simulations and have used the default values for Radio Duty Cycling (RDC) in Cooja: the contikimac_driver RDC driver with a channel check value of 8 Hz.

We show the different performance measurements gathered with the Cooja simulator to compare LO-CoAP-EAP performance with CoAP-EAP and PANATIKI. The evaluation is done in different scenarios: (1) with different numbers of hops ranging from 1–4 and (2) with different lossy environments with loss ratios of 0.0 0.1 and 0.2. These data have been gathered after performing around 100 authentications per scenario.

Table 4.3 Cooja test-bed specifications.

CPU	Intel(R) CoRE (TM) i5-2400 CPU @ 3.10 GHz
RAM	4 GiB DIMM DDR3 Synchronous 1333 MHz
O.S.	Ubuntu Server 12.04.5 LTS-32 bits
Kernel	3.13.0-32-generic

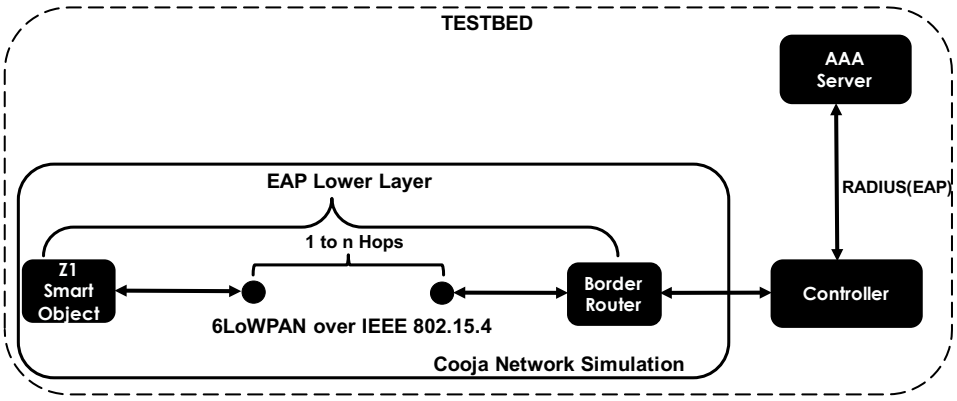


Fig. 4.5 Cooja test-bed.

Network Authentication Time in Cooja

Figure 4.6 shows the bootstrapping time. Generally, we can see that PANATIKI sets an upper bound to the authentication time, while LO-CoAP-EAP with handshake sets the lower bound. All CoAP-based solutions show a statistically significant different with respect to PANATIKI.

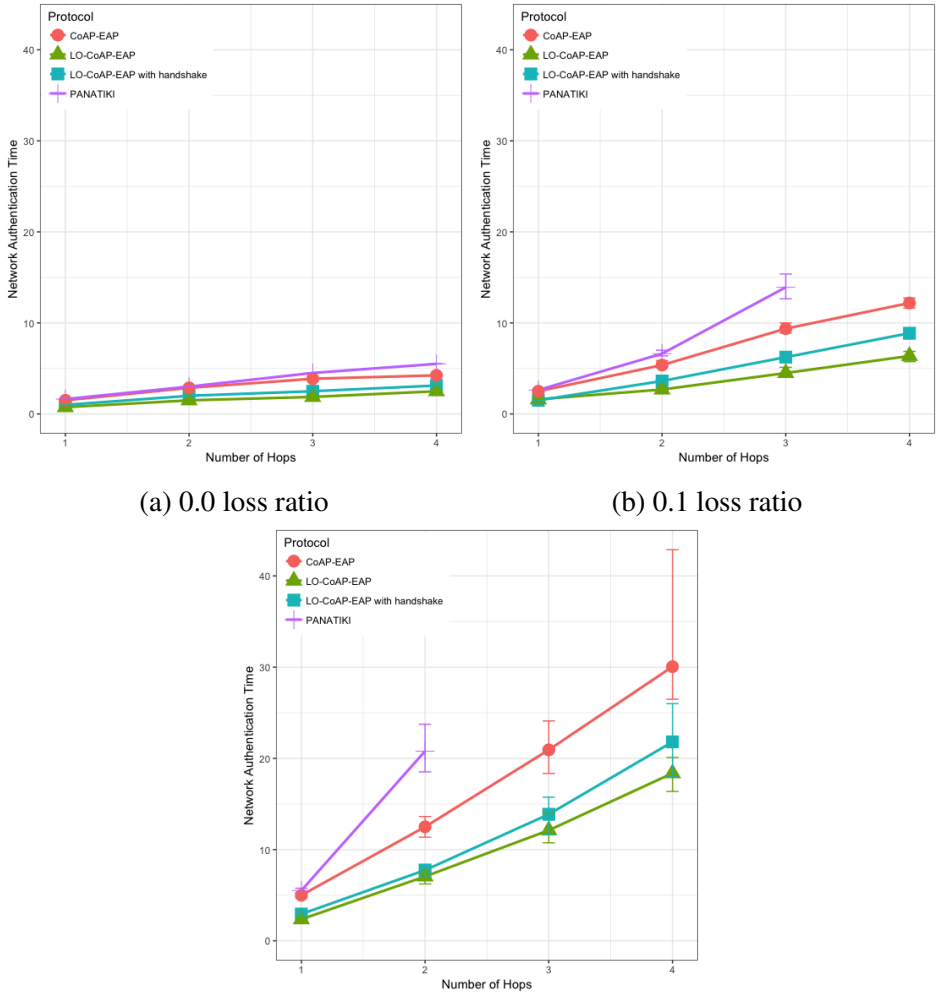
As we can see in Table 4.4, the improvement is up to ≈68% comparing any CoAP-based solution with PANATIKI. As expected, this difference is partially due to the reduction in message length of the CoAP-based solutions and the reduction of the number of messages in LO-CoAP-EAP. Since PANA has longer messages, PANATIKI takes longer to complete an authentication than any CoAP-based solutions. Among the CoAP based solutions, LO-CoAP-EAP improvement ranges from ≈26%–≈53% with respect to CoAP-EAP, also due to the reduction in size and number of exchanges in LO-CoAP-EAP.

With CoAP-EAP, there was little difference with PANATIKI in the most favorable conditions (fewer hops and loss ratio), whereas with LO-CoAP-EAP with or without handshake, the improvement is noticeable, even with more favorable conditions.

Figure 4.7 shows the success ratio with different loss ratios: 0.0 4.7a, 0.1 4.7b and 0.2 4.7c. When the packet loss ratio increases, the possibility of completing a bootstrapping procedure decreases. In this sense, we can see that PANATIKI sets a lower bound to the success ratio, while LO-CoAP-EAP sets an upper bound. CoAP-based solutions demonstrate a better performance in every packet loss ratio in comparison with PANATIKI. As mentioned before, the length and

Protocol	CoAP-EAP	LO-CoAP-EAP with Handshake	LO-CoAP-EAP without Handshake
PANATIKI	**4.5–39.9%**	**33.6–62.8%**	**38.8–67.7%**
CoAP-EAP	**-**	**26.5–41.3%**	**35.5–52.8%**
LO-CoAP-EAP with handshake	**-**	**-**	**0–28.2%**

Table 4.4 Comparing the improvement of network authentication Time among PANATIKI, CoAP-EAP, LO-CoAP-EAP without handshake and LO-CoAP-EAP with handshake.

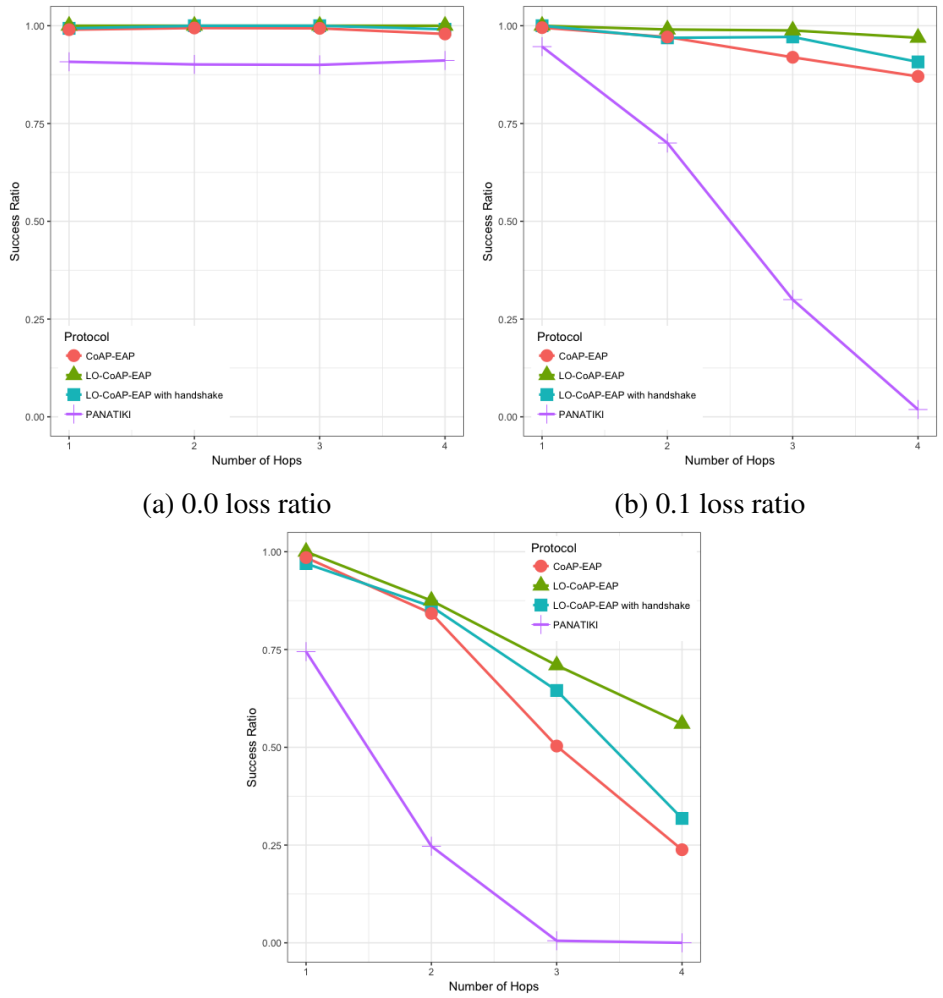


(a) 0.0 loss ratio

(b) 0.1 loss ratio

(c) 0.2 loss ratio

Fig. 4.6 Median network authentication time in Cooja for LO-CoAP-EAP, CoAP-EAP and PANATIKI



(a) 0.0 loss ratio

(b) 0.1 loss ratio

(c) 0.2 loss ratio

Fig. 4.7 Success Ratio in Cooja for LO-CoAP-EAP, CoAP-EAP and PANATIKI

quantity of messages to exchange play an important role, not only in the time it takes to complete an authentication, but also to complete the authentication successfully.

As can be seen, PANATIKI is not able to finish authentications with three hops. CoAP-based solutions are able to complete the authentication generally with a greater success ratio, since the reduction in the number of messages and their shorter message length has an impact on the overall number of exchanges and a reduction in fragmentation. Among the CoAP-based solutions, in the more favorable cases, the improvement is negligible and increases as the conditions are more severe. LO-CoAP-EAP improvement with respect to CoAP-EAP goes up to 43%. This difference can also be attributed to the reduction in message length and the number of messages.

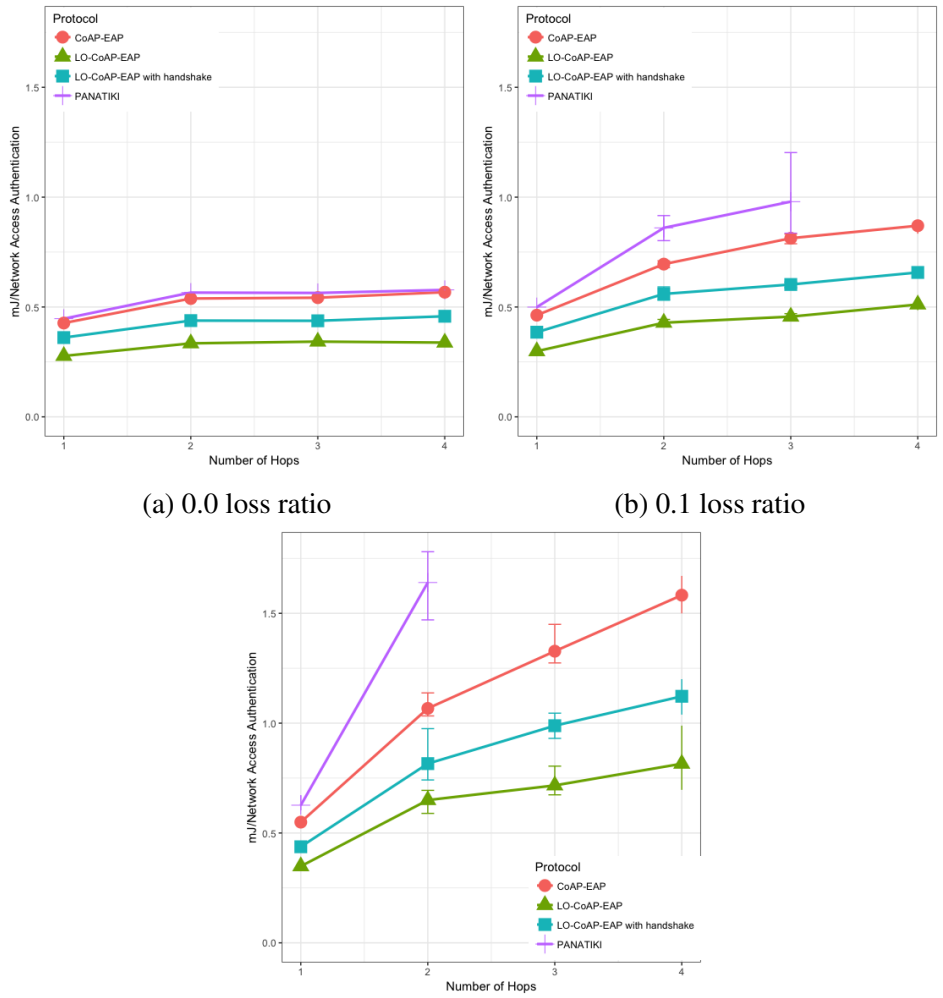
When evaluating the time to complete the bootstrapping, it is worth noting that this process is done prior to the smart object being able to send or receive data traffic. This means that how much time it takes to finish the authentication is not so important. However, it is important to finish it even in harsh conditions. This is even more relevant in LPWAN where the time to transmit and receive messages is considerable. For example, with a 0.2 loss ratio and four hops, LO-CoAP-EAP takes ≈18 s to complete the bootstrapping. This time is reasonable because: (1) we are assuming a very constrained link; (2) this process is only done once, before the smart object can do anything else; (3) it will only be done again in case the device looses its state or it resets.

Energy Consumption in Cooja

To perform the evaluation of the energy consumption, we used the Powertrace [51] tool that comes with the Cooja simulator. We use it to estimate the median energy consumed by each network authentication (mJ/network authentication) in LO-CoAP-EAP and CoAP-EAP (PANATIKI is also set as the reference). There are different measurements: CPU consumption when the smart object is fully operative (not in low power or sleeping mode); the consumption when transmitting (TX), receiving (RX) and the total energy consumption. For the sake of simplicity, we show the total energy consumption per authentication as a representative measurement, since it gives a general view of the requirements of each solution in terms of energy consumption. The total energy consumption is measured for three different loss ratio scenarios; 0.0, 0.1 and 0.2, respectively; and for different hops ranging from 1–4 hops between the smart object and border router.

Figure 4.8 shows the energy consumption with different loss ratios: 0 4.8a, 0.1 4.8b and 0.2 4.8b. The energy consumption, is greatly affected by the energy dedicated to sending and receiving messages. As the number and length of the messages increase, the energy consumption increases, as well. This is aggravated by the fragmentation of the messages, the retransmissions, etc.

The improvement, as shown in Table 4.5, ranges from 7%–63% comparing any CoAP-based solution with PANTIKI. The greater the length of the messages, more bytes are sent over the network. This is worsened when fragmentation occurs in a message, hindering the completion of an authentication. In the particular case of PANATIKI, it also has a more aggressive retransmission policy than CoAP, which results in sending more traffic over a constrained network. Among the CoAP-based solutions, LO-CoAP-EAP improvement ranges from 23%–50% with respect to CoAP-EAP. The reduction in size and number of messages is also a factor in the reduction of the energy consumption of LO-CoAP-EAP over CoAP-EAP. This is also noticeable comparing LO-CoAP-EAP with and without handshake.



(a) 0.0 loss ratio

(b) 0.1 loss ratio

(c) 0.2 loss ratio

Fig. 4.8 Total Energy Consmuption in Cooja for LO-CoAP-EAP, CoAP-EAP and PANATIKI

The added exchange of the handshake also influences significantly the energy consumption, as can be appreciated in Figure 4.8.

Memory Footprint in Cooja

Table 4.6 shows the memory footprint of each implementation. First, we show for reference the memory footprint of an empty program, representing the memory use of the O.S. After that, we show the memory footprint of each solution that includes the empty program measurements. The two columns represent how much memory is used in both ROM and RAM. The empty program uses ≈20 kB of ROM and ≈3.4 kB of RAM; PANATIKI uses ≈47 kB of ROM and ≈6 kB of RAM; CoAP-EAP ≈47.5 kB of ROM and ≈5.5 kB of RAM; and LO-CoAP-EAP ≈48 kB of ROM and ≈5.5 kB of RAM. These values include the O.S.,

Protocol	CoAP-EAP	LO-CoAP-EAP with Handshake	LO-CoAP-EAP without Handshake
PANATIKI	**6.7–32.8%**	**35.3–54.7%**	**49.3–63.3%**
CoAP-EAP	-	**23.2–32.6%**	**40.4–49.5%**
LO-CoAP-EAP with handshake	-	-	**15.8–27.7%**

Table 4.5 Energy consumption comparison.

Implementation	ROM (Bytes)	RAM (Bytes)
Empty Program	20,505	3,410
PANATIKI	47,151	6,006
CoAP-EAP	47,601	5,484
LO-CoAP-EAP	48,019	5,484

Table 4.6 Memory footprint of the implementations in Cooja.

the necessary network modules, the EAP state machine, the EAP method and the EAP lower layer. For a fair comparison, we use the same EAP state machine and EAP method.

Comparing the results, we can say that all EAP lower layers have a similar memory footprint. In PANATIKI, the O.S. employs ≈43.5% of ROM and 57% of RAM. For both CoAP-EAP and LO-CoAP-EAP, the S.O represents ≈43% ROM and ≈62% of RAM. The differences between CoAP-EAP and CoAP-EAP are in terms of code, to handle the case where it is handshake or not. Even though the CoAP-based solutions use more memory dedicated to the code, this is not an issue, since by using CoAP, we are sharing a common library present in most IoT devices (a CoAP implementation). PANATIKI would have to add the CoAP library to the existing code to support CoAP services, and this is one of the advantages of using a CoAP-based EAP lower layer for IoT (see more details in [66]).

4.4.3 LoRaFabian Network Test-Bed

For the test with LPWAN, we use a real LPWAN deployment: the LoRa network in Rennes, France. A snapshot of part of the city of Rennes where the deployment is located is shown in Figure 4.9. The deployment consists of three antennas (LoRa base stations) from Kerlink covering a fair portion of the city. Two of the antennas are installed on the structures maintained by *Telediffusion de France* (TDF) (Telediffusion de France), the company that

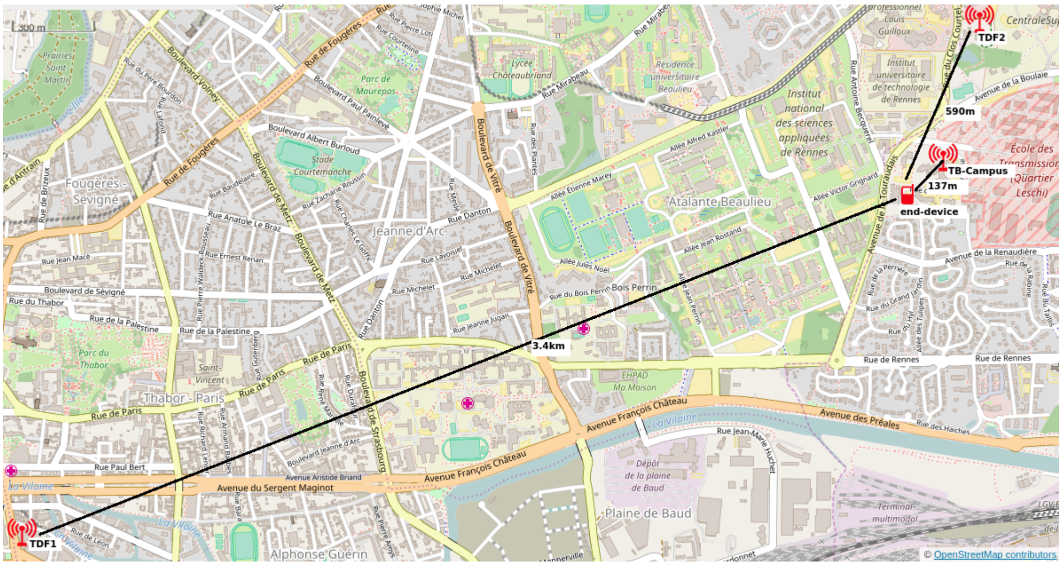


Fig. 4.9 LoRa Fabian network in Rennes, France.

provides telecommunication services in France, and the closest antenna is the one that is installed in the IMTatlantique's campus itself (previously known as Telecom Bretagne). The location of the measurement where the end device was used is more than 100 m away from the antenna in IMT atlantique.

The setup is shown in Figure 4.10. It has a star topology, which means the end nodes can reach the gateway in a single hop. The smart object instantiated in the end-device (from froggy factory (www.froggyfactory.com)) runs on Contiki and has an embedded LoRa radio coupled with an Arduino board to derive its power.

According to the architecture presented in Section 4.3.3, the controller will be in the gateway. Because it was not possible to avoid disrupting the production deployment, the controller was located in an external entity. For the gateway to communicate with the external authenticator, the CoAP messages were sent over HTTP to the controller that has a Python hook enabled that serves as a proxy to transfer the CoAP packets from HTTP to UDP. Although separated, in this test-bed, the controller and gateway would be co-located in the same entity in a production deployment. Finally, the AAA server runs FreeRADIUS Version 2.0.2 (freeradius.org).

In this test-bed, the nearest antenna is located <100 m from the end-device. These data have been gathered after performing 15 authentications per parameter to obtain bootstrapping time and energy consumption.

The LoraServer sends a beacon every 30 s, which is broadcast to the network using the antenna. The end device, upon receiving the beacon from the antenna, shall register itself to the network by sending a response to the beacon with its hardware address after a random

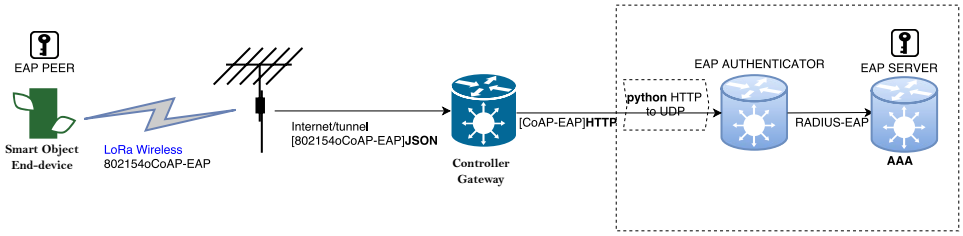


Fig. 4.10 Architecture of the LoRaFabian network.

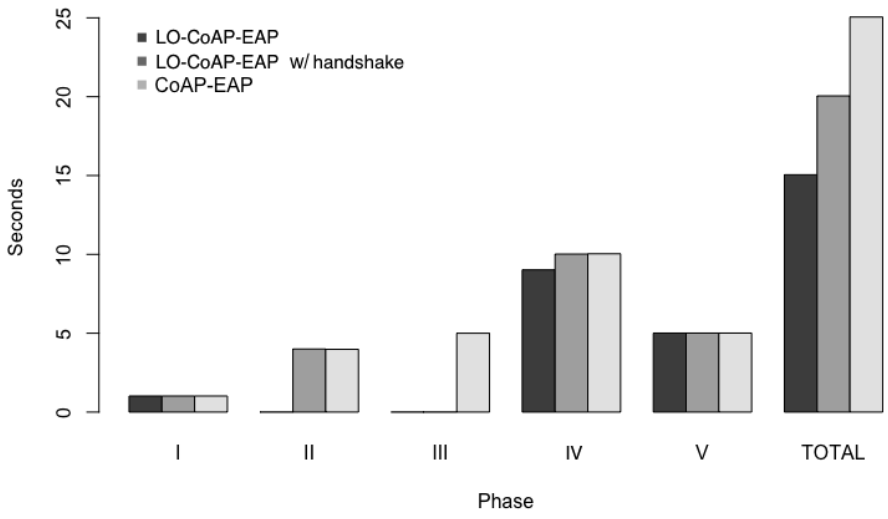


Fig. 4.11 Network authentication time.

delay in order to avoid collision with other devices. The network shall send a message to the devices by specifying the device's hardware address as the destination address in the 802.15.4 frame. The end device is continuously listening on the channel frequency except for the period of transmission.

Network Authentication Time in LoRa

A measurement that characterizes each solution of network authentication is the time it takes to complete. The graphs showing the median network authentication times can be seen in Figure 4.11. As may be appreciated in those figures, we can say that there is a statistically significant improvement in the network authentication time in LO-CoAP-EAP over CoAP-EAP.

To measure the network authentication, we have to consider that the Round Trip Time (RTT) constitutes the major portion of the bootstrapping time in this test-bed. It is the time

that is measured between two successive message received by the end device from the LoRa antenna, and it includes the travel time over the air, the message processing time in the Python hook, the authenticator and the RADIUS server. The total network authentication time is the interval that is measured between the time when the end device sends the first message to trigger the authentication mechanism (Phase I) and the time when the ACK for the last request containing the AUTH option is sent for the key confirmation (Phase V). Figure 4.11 shows the time taken by the end devices at each phase and the total time of the authentication process for one instance of each of the solutions (CoAP-EAP, LO-CoAP-EAP and LO-CoAP-EAP with handshake).

We can see that at Phase I, all solutions spend a similar time to complete this task. After that, CoAP-EAP and LO-CoAP-EAP with handshake show a similar time as a consequence of the handshake exchange. On the contrary, LO-CoAP-EAP saves this time. Regarding Phase III where the EAP identity is exchanged, only CoAP-EAP engages in that exchange. Phase IV, common to all solutions, where the EAP method is exchanged, presents similar values in the time spent during the exchange. Finally, for the final exchange, where the AUTH option is present, we can also see similar values for all solutions.

As we can see, LO-CoAP-EAP has fewer exchanges and less bytes traveling in the network, and for these reasons, it takes lesser time (15 s) to complete the bootstrapping as compared to the CoAP-EAP (25 s). LO-CoAP-EAP with handshake takes 20 s, an improvement of 5 s over CoAP-EAP, which is still a considerable improvement. Thus, we have achieved a reduction of $\approx$20% in the authentication time for LO-CoAP-EAP performing the handshake and a reduction of $\approx$40% in time for LO-CoAP-EAP. The question if this is a reasonable time has to take into account that this process is done one time, before the device can access the network. This is a necessary step to secure the communications and manage the network. Furthermore, the limited bandwidth in LPWAN (LoRa in this case) bound longer transmission times, so we think that these times are not unusual.

As a side note and apart from the differences with the Cooja simulations, we can see that we have similar values in LPWAN to the worst cases in Cooja (with a 0.2 loss ratio and 3–4 hops). This gives us an approximation of the harsh conditions LPWAN networks are dealing with to transmit data over the network.

Energy Consumption in LoRaFabian

As there is no Powertrace application support for the LoRa platform, we have used a digital multimeter to measure the current drawn by the shield containing the smart object, as in Figure 4.12 during the network authentication process.

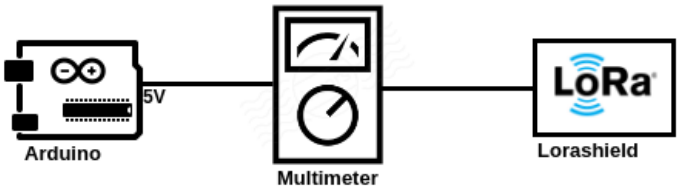


Fig. 4.12 Energy measurement setup.

Table 4.7 Energy measurement in LoRaFabian.

CoAP-EAP network authentication	2453.92 mJ/authentication
LO-CoAP-EAP (with handshake) network authentication	1964.9 mJ/authentication
LO-CoAP-EAP (without handshake) network authentication	1473.92 mJ/authentication

The smart object is in idle mode until it receives the first radio packet and continues to work in radio mode throughout the network authentication process.

The parameters used to measure the energy consumption are the following: the nominal power for operation is 5 V; the current consumption in each mode is: 15 mA when idle and 19.6 mA in transmission (TX) and reception (RX). With these values, we measure the energy consumption of each solution.

Table 4.7 shows the energy consumption of the test done in LoRa. Comparing the energy consumption of the solutions and taking as reference CoAP-EAP, we appreciate a reduction of ≈20% in energy consumption for LO-CoAP-EAP performing the handshake and a reduction up to ≈40% in energy consumption for LO-CoAP-EAP. This clearly shows a considerable improvement over the previous solution.

Memory Footprint in LoRaFabian

Table 4.8 shows the memory use for the implementations. For these experiments, we use the STM32F103RB mote, which as 128 kBytes of ROM and 20 kBytes of SRAM. For reference, we show the memory footprint of an empty program and after that the values of CoAP-EAP and LP-CoAP-EAP. The empty program uses ≈8 kB of ROM and ≈2.6 kB of RAM. Both CoAP-based solutions have a similar memory footprint; CoAP-EAP ≈40.7 kB of ROM and ≈8.7 kB of RAM; LO-CoAP-EAP ≈41.1 kB of ROM and ≈8.7 kB of RAM.

The empty program is an estimate of the memory footprint of the O.S. CoAP-EAP and LO-CoAP-EAP include, additionally, the necessary network modules, the EAP state machine, the EAP method and the EAP lower layer. The EAP state machine and EAP method are

Table 4.8 Memory footprint of the implementations in LoRa.

Implementation	**ROM (bytes)**	**RAM (bytes)**
Empty Program	7,908	2,596
CoAP-EAP	40,744	8,668
LO-CoAP-EAP	41,144	8,668

the same in all instances. Comparing the results, we can say that for both CoAP-EAP and LO-CoAP-EAP, the O.S. represents ≈20% ROM and ≈33% of RAM. The differences between CoAP-EAP and LO-CoAP-EAP are in terms of code, due to the changes to manage the situation with handshake.

4.5 Conclusions

In this chapter, we highlight how the incorporation of new radio technologies in the IoT landscape, known as LPWAN, poses an interesting challenge, when bootstrapping is taken into account. LPWAN s have even more restricted links than those known in LR-WPAN. In particular, besides the existing constraints in LR-WPAN, in terms of resources such as memory, CPU and energy consumption, the bandwidth is also severely restricted. We have presented LO-CoAP-EAP, a complete redesign of a previous solution for bootstrapping (CoAP-EAP), to cope with the restrictions imposed by LPWAN. LO-CoAP-EAP has been designed to further reduce the number and the length of messages to deal with the very limited LPWAN bandwidth. LO-CoAP-EAP still uses CoAP, EAP and AAA to provide scalability, flexibility and wireless independence, however reducing the number and length of the messages, although, once the bootstrapping is finished successfully, it is possible to provide key material to different types of LPWAN security associations to protect the access to the network. To assess the performance of LO-CoAP-EAP and to show how it overcomes previous work, we have performed simulations with the Cooja network simulator to confirm that LO-CoAP-EAP improves CoAP-EAP and PANA in the context of LR-WPAN. Not only that, we have also run LO-CoAP-EAP and CoAP-EAP over a real LoRa network, a LoRaFabian network, as a representative of LPWAN, to experimentally prove that the LO-CoAP-EAP provides an improvement in both LPWAN and LR-WPAN networks. We have obtained an improvement in network authentication time of up to 53% in Cooja and an improvement of 20% to 40% in the real LPWAN (LoRaFabian). We achieved a reduction in energy consumption from 11% to 46% in Cooja and a reduction of up to 40% in LoRaFabian.

In the next chapter, we consider the possibility of having to perform the bootstrapping in a mesh network where the smart object is not able to reach the Controller because it lacks routable IP and needs the assistance of another entity that aids the smart object in the bootstrapping process.

Chapter 5

Multi-hop bootstrapping through CoAP intermediaries for IoT

This chapter introduces the last of the main contributions of this thesis: *support for bootstrapping in multi-hop networks through CoAP intermediaries*. In this chapter, we explain the need for an intermediary entity in a multi-hop network, when a smart object trying to join the network cannot reach the Controller by its own means. We show the design of three intermediary entities based on CoAP : a CoAP proxy, a CoAP relay and a CoAP stateless proxy. We also also show the extended architecture of the bootstrapping service, including the intermediary entity, as well as the flow of operation of each alternative. After that, we present the experimental results and compare each alternative as well as the performance of the other EAP lower layer for the IoT with an intermediary entity, the PANA relay.

5.1 Introduction

The global network of Internet-connected objects known as the *Internet of Things* (IoT) is growing at a fast pace [75]. This great volume of devices with built-in capabilities to access the Internet has to be securely managed throughout their life cycle [69]. This management includes the *bootstrapping* process, which implies the authentication, authorization and key distribution required to allow a smart object to securely join a network.

The previous proposals assume that the Smart Object is able to reach the Controller to perform the bootstrapping. But there are scenarios in which the *Smart Object* joining the network is not able to communicate with the *Controller*. This may be, for example, the result of being farther from the expected radio distance or, at the network layer, the Smart Object may lack of a routable IP address, which is only configured after the successful

bootstrapping. This can be solved with an intermediary that aids the Smart Object in the process by providing support in multi-hop networks. For example, the Zigbee IP standard [217] uses PANA [61] to transport EAP [10] for bootstrapping. The *PANA relay* [50] is the intermediary entity in PANA to assist this process. Alternatively, using CoAP [190] for this purpose is being considered in groups like the *6TiSCH* IETF WG. The reason is that CoAP is a lightweight protocol specially designed for constrained devices and networks.

As with PANA, we have already considered the use of EAP for bootstrapping in previous work but considering CoAP as the protocol to steer the bootstrapping. Our solution, named LO-CoAP-EAP [67] [66] improves PANA, due mainly to the use of CoAP as a transport for EAP . However, LO-CoAP-EAP does not define any intermediary to aid the Smart Object in contacting the Controller. In order to fill this gap, we now extend LO-CoAP-EAP with an intermediary.

Specifically, we analyze, design and evaluate three alternatives for the LO-CoAP-EAP intermediary: 1) the LO-CoAP-EAP proxy, 2) the LO-CoAP-EAP relay and 3) LO-CoAP-EAP stateless proxy. The LO-CoAP-EAP proxy is based on the CoAP proxy, defined in the standard, but the CoAP relay and stateless proxy are not, which we do in this chapter for the bootstrapping case with LO-CoAP-EAP. The LO-CoAP-EAP proxy is stateful and can analyze and modify the original LO-CoAP-EAP messages. The LO-CoAP-EAP relay keeps the intermediary stateless by tunneling the LO-CoAP-EAP messages between the Smart Object and the Controller within other CoAP messages. Finally, the LO-CoAP-EAP stateless proxy is a hybrid solution between the relay and the proxy since it does not keep any specific state per Smart Object, but can modify the original messages between both the Smart Object and the Controller. For comparison, we have implemented a proof-of-concept of each intermediary in Contiki O.S. [52], evaluating their performance using the Cooja simulator [146] and comparing them with the PANA relay, as the existing standard deployed in IoT networks. It is worth noting that the design and implementation of the intermediaries are independent of the authentication protocol deployed.

The rest of this chapter is divided as follows. Section 5.2 describes the state of the art. In Section 5.3 we present the extended architecture, the design of the LO-CoAP-EAP relay and the LO-CoAP-EAP proxy and stateless proxy. In Section 5.4, we show the experimental results and the comparison with the PANA relay. Finally, we draw the conclusions and future work of this research in Section 5.5.

5.2 Related Work

As explained by Garcia-Morchon *et al.* [69], a node needs to perform what is called the *bootstrapping* to join a network and become part of the security domain. This process entails authentication and authorization, as well as obtaining the necessary key material and configuration parameters, e.g., global IP address, to begin its activity as an authenticated party in the domain.

For example, the standard Zigbee IP [217] defines IPv6-based wireless mesh networks for remote control and sensing, with an interoperable protocol stack using networking protocols defined in the IETF and the standard IEEE 802.15.4 [7]. For bootstrapping purposes, they use the *Protocol for Carrying Authentication and Network Access* (PANA) [61]. The node joining the network is the *PANA client* (PaC), also called *joining node*. The *Zigbee IP Coordinator* is the *PANA Agent* (PAA), acting as Controller's domain. Having a multi-hop topology, Zigbee IP requires an intermediate entity, and they have defined the *PANA relay* (PRE) [50]. The role of PRE is assigned to the parent of the *joining node* unless the parent node is the *PAA*. Regarding the operation of the PRE, it appears to the PaC as if it were the PAA.

While solutions that use EAP [67] are, generally, considered for large scale scenarios since it may rely on AAA infrastructures, other solutions for smaller scales also include an intermediary entity to aid in the bootstraping. For example, the IETF 6TiSCH WG [95] describes in [206] the process that allows a Smart Object (named *pledge*) how to join the network. To control the access, JRC acts as the Controller's domain, managing authentication, authorization and distributing key material. They also consider a generic entity called JP, which is a 1-hop radio neighbour to the *pledge* and helps it to communicate with the JRC for the joining procedure. In essence, the JP is a CoAP proxy but with a new CoAP option to enable a stateless proxy operation. Unlike LO-CoAP-EAP, the authentication assumes the pledge implements the CoAP client and the Controller the CoAP server and, as a consequence, the pledge implements a re-trantransmission mechanism. This model is not applicable when EAP is used as authentication protocolo for the bootstrapping and CoAP is used as EAP lower-layer, as explained in section 3.6.1 in [66]. The reason is that EAP sends requests from the Controller to the Smart Object and receive EAP response from the Smart Object. Therefore, with EAP, the Controller must be the entity steering the communication and not the Smart Object. Moreover, the exchanges are based on NON confirmable messages to reduce the state stored in the message layer while our solution is based on CON messages. The main reason is that to avoid any further modification in Smart Objects that already implement LO-CoAP-EAP. In fact, a design principle in our solution is to keep unmodified as much as possible the Smart object by the inclusion an Intermediary. In other words, practically the same implementation should work with or without intermediary. Therefore

Intermediary	Auth. Protocol	Auth. Credential	Smart Object/ Controller SA
6TiSCH stateless proxy[206]	OSCORE	PSK	OSCORE
DTLS Relay (stateful and stateless) [108]	DTLS	PSK, Certificates, raw public keys	DTLS SA
PANA relay (stateless) [50]	EAP	Flexible	PANA SA (integrity only)
LO-CoAP-EAP [67] relay, proxy and stateless proxy (these intermediaries are defined in this article)	EAP	Flexible	AUTH SA (integrity only)

Table 5.1 Summary of existing intermediaries. (*SA: Security Association*)

CON messages are used. Moreover the design of an EAP lower-layer based on CoAP requires CON so that the Controller manages the res-transmissions and ACK with piggybacks to reduce the number of messages (see Fig. 4.1).

Specifically, the 6TiSCH configuration requires that the pledge and the JRC share a symmetric key (PSK), local to the domain. However, how this key is provisioned is out of scope of the specification. In this sense, LO-CoAP-EAP can assist by generating dynamically the 6TiSCH PSK, with the help of the intermediaries defined in this chapter. Therefore, LO-CoAP-EAP with intermediaries is compatible with the posterior usage of the solution in 6TiSCH. Similary, the *Datagram Transport Layer Security* (DTLS) relay (stateless and stateful) by S. Kumar *et al.* [108] assists a DTLS client that is not authenticated and has no routable IP to authenticate with the DTLS server. The stateless DTLS relay is based on PANA relay, encapsulating DTLS messages in a *DTLS Relay* message. While the stateful DTLS relay is based on the concept of proxy by keeping track of the whole DTLS negotiation as it were the DTLS client.

Finally, it is worth noting that when the authentication is based on EAP (e.g. either PANA or LO-CoAP-EAP), the cryptographic material (e.g. MSK) can be used to bootstrap a pre-shared key (PSK) to run the DTLS security association. Therefore, when DTLS starts, the smart object is already authenticated, and the DTLS relay is not needed to assist in the joining phase.

Table 5.1 show a summary of the different intermediaries found in the literature. We show the *authentication protocol* used during the bootstrapping, the type of credentials for the authentication (pre-shared keys, certificates, etc..) and the security association (SA) established after the bootstrapping between the Smart Object and the Controller.

5.3 LO-CoAP-EAP Bootstrapping with Intermediary

With the deployment of intermediaries, there are several considerations that need to be taken into account when it comes to bootstrapping process, namely:

- The messages sent over the network between the Intermediary and the Controller will impose an additional overhead over the original messages.
- The traffic with the Intermediary is not secured when the Smart Object is not authenticated. The reason is they do not share any key material to establish a security association. On the contrary, it is assumed that the Intermediary is an authenticated entity in the Controller's domain (e.g. it performed a bootstrapping process) and has an active security association with the Controller.
- To avoid misuse of the network resources, the Intermediary will only allow traffic from unauthenticated entities related to the bootstrapping service (filtering out otherwise), sending it only to the Controller. All traffic related to the authentication service is protected between the Intermediary and the Controller with their shared key material. A security association such as OSCORE [185] or DTLS may be used for this purpose.
- At first instance, the Intermediary appears to the Smart Object as if it were the Controller. In other words, the Smart Object does not need to know whether a 1-hop neighbour is actually the Controller or an Intermediary until the end of the authentication. On the contrary, the Controller is aware that there is an Intermediary involved during the authentication process.
- The Smart Object will start the bootstrapping exchange once the layer 3 is setup (e.g. IPv6 link-local address is ready to be used). The first message is treated as a special case and it is used as a trigger regardless of the type on intermediary (relay, proxy or stateless proxy), avoiding the creation of any initial state in the intermediaries. The exchange from the Smart Object's perspective, will be the same as the one described in Chapter 4, except in the last exchange, where the Controller informs about the presence of an Intermediary in a secure fashion.

With these considerations, we present the extension to the LO-CoAP-EAP architecture with the Intermediary and we go through each of the alternatives we have considered in this chapter.

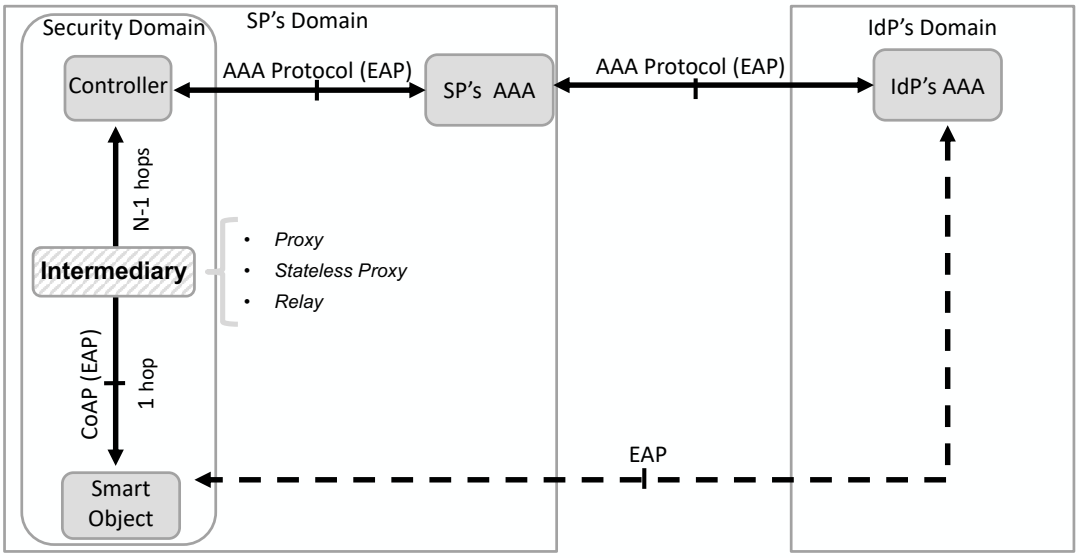


Fig. 5.1 LO-CoAP-EAP multi-hop architecture

5.3.1 LO-CoAP-EAP architecture with an Intermediary

We have extended the LO-CoAP-EAP architecture including an intermediary entity to aid newcomers in the bootstrapping. The extended LO-CoAP-EAP architecture is illustrated in Fig. 5.1. The logical placement of the *Intermediary* is between the *Controller* and the *Smart Object*. The Intermediary is a 1-hop neighbour of the Smart Object, and one or more hops away from the Controller.

Concretely, we have analysed three entities to fulfill the intermediary role: a *relay* that tunnels the CoAP messages received from the Smart Object to the Controller and vice versa using a new CoAP message, which makes the intermediary stateless; a *proxy*, such as defined in CoAP standard, which can modify the original messages between the Smart Object and the Controller but needs to keep state for the operation; a *stateless proxy* that is an hybrid solution where the Intermediary is stateless in the sense it does not keep per Smart Object state, but it can still modify the CoAP messages between the Smart Object and the Controller.

Without loss of generality, we use the bootstrapping based on *LO-CoAP-EAP without bootstrapping* for the explanation. Moreover, although LO-CoAP-EAP is independent of the EAP method, we use EAP-PSK [25] to describe the interactions for simplicity. EAP-PSK is an authentication mechanism that serves as a representative since it offers a lightweight authentication mechanism. [1]

[1] The PSK in EAP-PSK must not be confused with 6TiSCH PSK. PSK in EAP-PSK is considered a long-term credential since it is set during Smart Object's commissioning. On the contrary, 6TiSCH PSK is

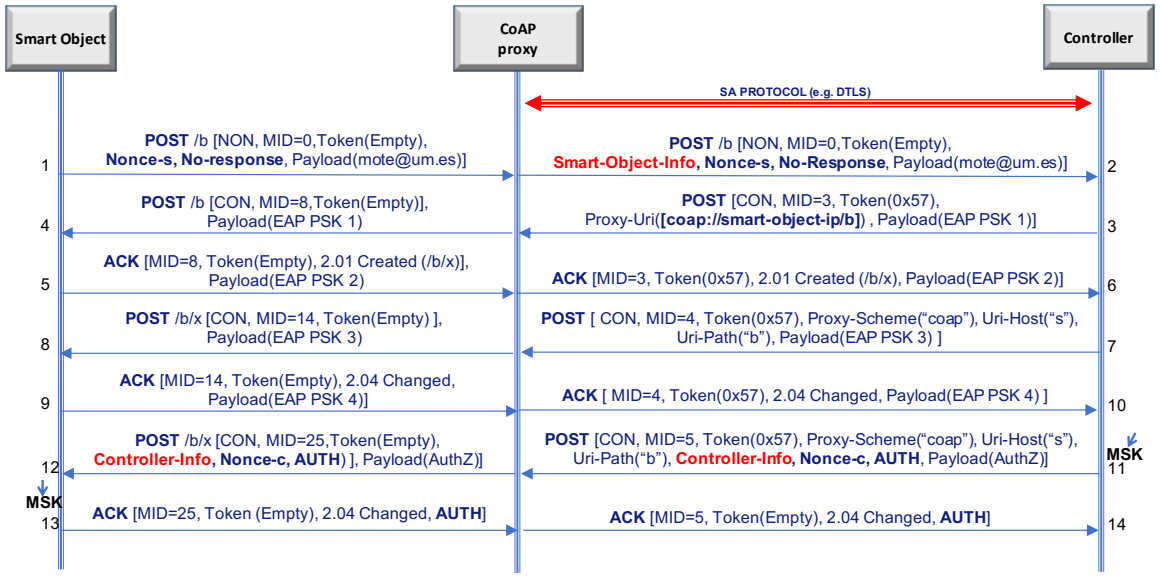


Fig. 5.2 LO-CoAP-EAP proxy

5.3.2 LO-CoAP-EAP proxy

The design of the LO-CoAP-EAP proxy is based on the CoAP proxy [190] defined in the CoAP standard. Concretely, the LO-CoAP-EAP proxy must be configured as a forward-proxy that does not use a cache. Nevertheless, it keeps a state related to the ongoing exchange between the Controller (CoAP client) and Smart Object (CoAP server) in order to reduce the number of bytes sent over the network.

Figure 5.2 shows how the network authentication process occurs with the involvement of a LO-CoAP-EAP proxy. The Smart Object sends the initial message to the proxy (step **1**). As we can observe, the resource */b* is the one for the bootstrapping. As such, the proxy is able to process and parse the request to that resource as part of the bootstrapping service. The initial message also contains the Smart Object's identity expressed in *Network Access Identifier* (NAI) format [46] (e.g. **mote@um.es**). The proxy adds the Smart Object's IP to the message in a new defined *Smart-Object-Info* Option (step **2**) and forwards the message to the Controller. The *Smart-Object-Info* Option carries an IP address and (optionally) an UDP port. The UDP port might be required when the Smart Object is not operating in the default port for a CoAP server (*5683*). At this point no state is reserved. It is the next message from the Controller (step **3**) that creates the state in the proxy. In this message, the Controller starts the EAP authentication communicating with the Smart Object through the proxy. It sets the *Proxy-Uri* option with the information gathered from the *Smart-Object-Info* Option

considered a local session key dynamically generated from the MSK in the domain where the bootstrapping has happened.

from the previous message. It also sets a Token with an unique identifier of the ongoing authentication, associating the Token's value with the information (Smart Object's IP and Port) contained in the *Proxy-Uri*.

As we defined in [66] the Token value is used as session identifier for the authentication. Then, the proxy creates state where it stores the Token value (0x57 in the example) as authentication session together with Smart Object's IP and the resource created (*x*) by the Smart Object in step **5**). As a consequence, the following POST (Steps 7 and 11) only carries the token (0x57), the Proxy-Schema("coap"), *Uri-Host("s")* with a value defined in this article, the name *"s"*, and *Uri-Path ("b")*. Since the proxy has stored the Smart Object's IP in Step **3**, the *Uri-Host("s")* tells the proxy that uses the IP stored to reach the Smart Object, so that there is no need to resolve the name *"s"*. Moreover, just specifying *Uri-Path ("b")* is enough since the proxy already knows that the path information is */b/x* due to Step **5**. This avoids sending the complete Proxy-Uri Option repeatedly again to the proxy. In other words, we leverage the state created in the proxy to save bytes over the network.

Afterwards, the EAP conversation continues as expected (steps **3**-**10**). Once the EAP authentication is completed, the Controller receives the MSK from the EAP server and sends the last message to the Smart Object. In this message (steps **11**-**12**) the Smart Object is aware that it was talking with the Controller through an Intermediary; it finds the Controller's IP in a new option named *Controller-Info*, which carries an IP address and (optionally) a UDP port. This last exchange (step **11**-**14**) is protected with integrity between the Smart Object and the Controller (using the AUTH option defined in [66]) but only the content of the message that is not susceptible to be modified by the proxy, similarly to OSCORE [185][2]. After the authentication, the Smart Object and the Controller can perform a *ch6:security Association Protocol* (SAP) (see Section 5.3.5).

It is worth noting that the Controller can choose a MID different than the proxy when sending a request to the Smart Object (e.g. in Step 2 the Controller has MID=3 and proxy chooses MID=8). The reason is that the proxy stores the relationship between the MID from the Controller (MID=3) and the one expected (MID=8). Once the proxy receives the message from the Smart Object (with MID=8) can map this value to the MID expected by the Controller (MID=3). Table 5.2 summarizes how the Intermediaries use the MID and Token values.

[2] It is wort noting that we defined AUTH previously to the definition of OSCORE

	MID	Token
Proxy	The Controller chooses its own MID. The Proxy stores the Controller's value and chooses its own value *Message ID* (MID)	The Controller chooses a Token value that the Proxy will store as authentication session identifier. The Proxy will use empty value with the Smart Object since there can be only an ongoing bootstrapping with the Smart Object [67]
Relay	The Controller chooses the MID for the CoAP messages inside the CoAP-based tunnel. The MID is 0 for the LO-CoAP-EAP relay messages to avoid storing any state.	The Token value is empty to avoid keeping state for this value.
Stateless Proxy	The Controller chooses the MID for the CoAP messages. The Stateless Proxy uses the same value than the one chosen by the Controller.	The Token value is empty to avoid keeping state for this value.

Table 5.2 Summary of the usage of Message ID (MID) and Token in the proxy, relay and stateless proxy operation.

5.3.3 LO-CoAP-EAP relay

Unlike the CoAP proxy, the *CoAP relay* is not defined in the CoAP standard. We propose a design for LO-CoAP-EAP relay as the simplest alternative for an intermediary. To achieve simplicity, the LO-CoAP-EAP relay has been designed with two important features: 1) it does not need to manipulate or analyze the messages from the Smart Object or the Controller and, 2) it is completely stateless so it does not store any information related to the exchange.

To achieve the first feature, the design principle is using the concept of encapsulation by using a CoAP-based tunnel between the relay and the Controller. To accomplish the second, the relay includes the information related to the state (i.e., Smart Object's IP and port) into each CoAP message used for encapsulation. The Controller also sends this state back to the relay again so the relay knows where to send the message. We have defined the CoAP message for encapsulation (*CoAP relay message*) as follows: it is Non-confirmable (NON) to avoid retransmission and the ACKnowledgement message; it contains the *No-Response* option [26] to avoid any related response to this message; the *Message ID* is always set to 0 (MID=0), since it does not have to detect message duplication or match message types. Similarly, the *Token* is set to empty since there is no need to match a response with a request since there is no response at the CoAP relay messages. Thus, re-transmissions must be completely handled by the Controller, which is the entity steering the LO-CoAP-EAP bootstrapping. Finally, both the relay and the Controller must implement a resource with the *URI-Path /r* to process the *CoAP relay message*.

Figure 5.3 shows the flow operation. The Smart Object sends the first message (step **1**) to trigger the authentication process. When the relay receives that message and, since it is a message to the resource $/b$, it creates a *CoAP relay message* and encapsulates the message

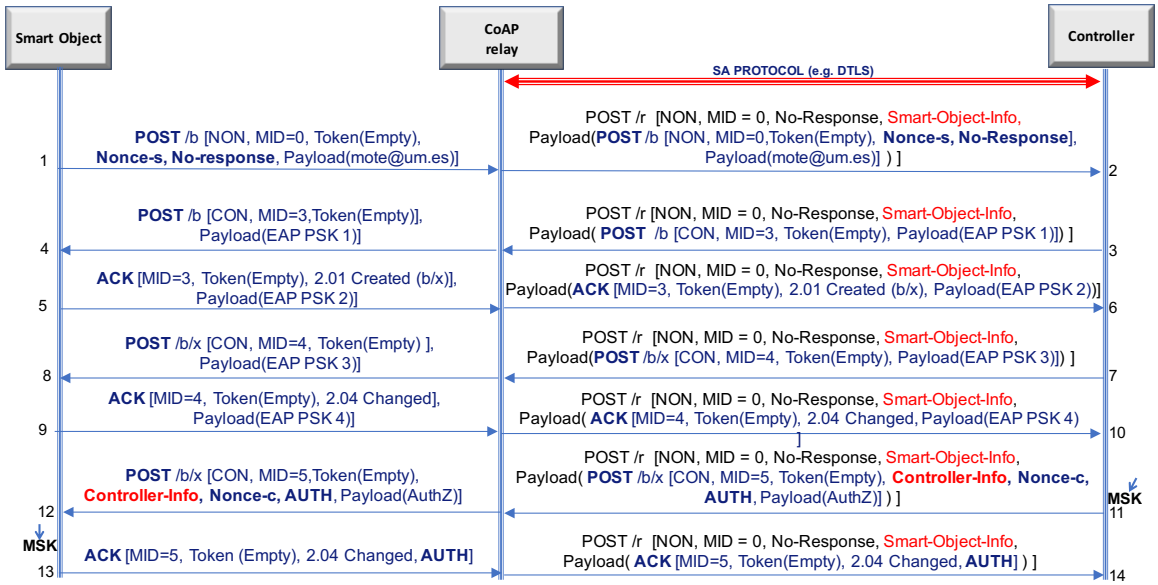


Fig. 5.3 LO-CoAP-EAP relay

(step **2**) and sends it to the Controller. When the Controller processes the trigger, it starts the bootstrapping. It decides to perform a bootstrapping based on LO-CoAP-EAP *without handshake* as shown in Figure 5.3) or *with handshake* with the Smart Object. In any case, the Controller encapsulates the LO-CoAP-EAP message into a *CoAP relay message* with the Smart Object's IP and (optionally) UDP port (*Smart-Object-Info* option) and sending them to the LO-CoAP-EAP relay (step **3**). The relay then gets the content of the message and, specially, the Smart-Object-Info, and relays the message to the right Smart Object (step **4**). The general flow continues with the same process: encapsulating LO-CoAP-EAP messages into *CoAP relay messages* (steps **5** to **14**).

As we may observe, the Controller includes its own IP information in the request of the last exchange (*Controller-Info* Option in step **11**) so that the Smart Object is able to know it was speaking through an Intermediary. Therefore, the key material obtained from the authentication is to be used to establish a security association with the Controller.

It is also important to note that, as mentioned in [66], the Controller needs to handle different MIDs for different Smart Objects. Therefore, the Controller sends a *Message ID* (MID) (e.g. MID=3 in step **3**), and this MID is used for a particular Smart Object. The relay only forwards the encapsulated message and, therefore, it does not handle this value when forwarding the message to the Smart Object (step **4**). In this manner, when the Smart Object returns the answer (step **5**), it will use the same the MID that the Controller set. Therefore, this value will be recognized as valid by the Controller. Table 5.2 summarizes the values used during the exchange in Fig. 5.3 in the column *Relay*.

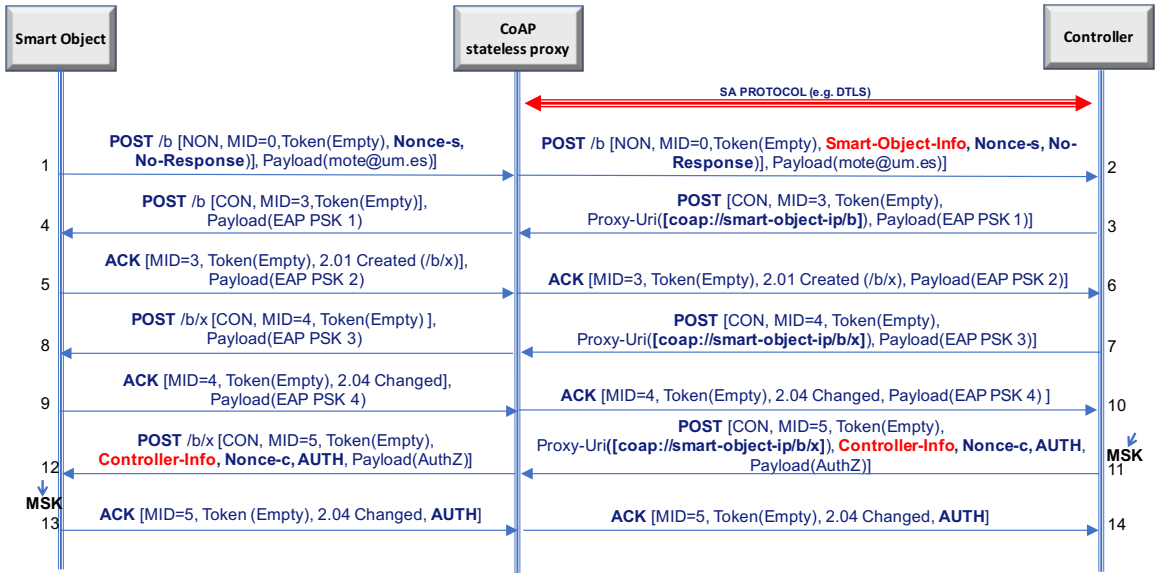


Fig. 5.4 LO-CoAP-EAP stateless proxy

5.3.4 LO-CoAP-EAP stateless proxy

The LO-CoAP-EAP stateless proxy provides a hybrid solution and a tradeoff between the LO-CoAP-EAP proxy and the relay. That is, it does not need to store *per-Smart Object* state as the proxy does, alleviating the possibility of Denial-of-Service (DoS) attack (as it may happen with the Proxy, which stores per-Smart Object state) and it avoids the message size increment as a consequence of the encapsulation between the relay and the Controller. The stateless proxy acts as a CoAP proxy for the Controller but it acts as a relay when communicating with the Smart Object. This design achieves several goals. First, it avoids that the stateless proxy stores any state for re-transmission in the communication with the Smart Object; the Controller will be completely in charge of handling re-transmission timer and associated state.

To achieve this goal, it avoids keeping any specific state about the recipient Smart Object of the messages coming from the Controller, at the cost of sending this information on each message sent by the Controller. Finally, the stateless proxy saves bytes over the network compared to the relay because encapsulation is not required with the Controller.

The exchange of the LO-CoAP-EAP stateless proxy is shown in Fig. 5.4. As occurs with the LO-CoAP-EAP proxy, the Smart Object sends the first message (step **1**) and the stateless proxy adds the Smart Object's IP (and optionally UDP port) to the request in the *Smart-Object-Info* option, forwarding it to the Controller (step **2**). From this point on, the Controller uses the LO-CoAP-EAP stateless proxy as a forward proxy in each request. Unlike the LO-CoAP-EAP proxy, the Controller sets the Token value to empty. The reason is that

the stateless proxy will not store this value during the conversation and therefore it is useless to send it because the stateless proxy will not return it back. In contrast, the Controller uses *Proxy-Uri* option on every message with the information that the stateless proxy needs to know about the Smart Object to forward the message to the right one. This implies additional bytes over the network. It is also important to note that when the Controller sends a *Message ID* (MID) (e.g. MID=3 in step **3**), instead of storing this value, the stateless proxy will maintain this MID when forwarding the message to the Smart Object (step **4**). In this manner, when the Smart Object returns the answer (step **5**), it will use the same the MID that the Controller set. Therefore, this value will be recognized as valid by the Controller when the stateless proxy forwards the answer coming from the Smart Object (step **6**) and, therefore, the stateless proxy does not need to store this value, as happened with the proxy. This is possible because the Controller knows what MID used for a particular Smart Object (the Controller will not use overlapping MIDs with different Smart Objects), so based on that value it can track when it receives a ACK from the stateless proxy what Smart Object answered. Table 5.2 summarizes the values used during the exchange in Fig. 5.4 in the column *Stateless Proxy*. The LO-CoAP-EAP exchange continues (steps **7-14**) until complete the authentication. Since Smart Object is not aware of the Controller, it indicates its own IP (step **11**) to the Smart Object (*Controller-Info*).

5.3.5 Establishing a Security Association after bootstrapping

After a process of bootstrapping the Smart Object and the Controller can perform a Security Association Protocol (SAP) to protect further communications between both entities. This is possible thanks to a key derived from the recently established MSK. As an example, this SAP can be the join procedure defined in 6TiSCH or running DTLS (see Section 5.2). Additionally, it is important to note that the Smart Object will have to establish a SA with the Intermediary to secure their communications. This SA can be unicast (e.g. *link keys* in Zigbee) or multicast (e.g. *network key* in Zigbee) or both. The distribution of the key material to establish these SAs can use the unicast SA between the Smart Object and the Controller as starting point for a secure key distribution. For example, one may consider the distribution of multicast keys encrypted from the Controller to the Smart Object after the bootstrapping is performed (e.g. this a model observed in Zigbee). Alternatively, the Controller may distribute an unicast key for the Intermediary and the Smart Object, levering the Controller-Intermediary unicast SA and the recently established Controller-Smart Object unicast SA.

Since our solution for bootstrapping is independent of the technology, analyzing the different models for key distribution within Controller's security domain just after the bootstrapping is considered as part of the future work.

5.4 Experimental Results

To carry out a performance evaluation, we have prepared a testbed with the Cooja Network Simulator for Contiki OS, version 2.7 [146]. Between the Intermediary and the Border Router/Controller there can be 1 hop (direct link), or other smart objects acting as IP-forwarders. Each scenario is run with different loss ratio values, namely: 0.05, 0.1 and 0.15; and different number of hops to simulate cases with different network conditions. Using 0.0 loss ratio is an ideal scenario, but not very realistic. With 0.2 loss ratio we found that the network was so deteriorated that there was no clear distinction between the CoAP-based intermediaries, and the PANA relay was unable to complete an authentication beyond 2 hops. We also use different number of hops to range a different cases from a Smart Object directly connected to the Controller to the case of several hops (*6*) to reach the Controller. This number has been defined so because beyond that, any bootstrapping was able to finish properly due to re-transmissions exceed the maximum number. The smart objects used for this test-bed are Zolertia Z1 with 92kB of nominal ROM, compiled with 20-bit architecture support and 8kB of RAM. The compiler is the *msp430-gcc* version 4.7.2. To communicate the simulation with the outside physical network (where the Controller is placed) we use the RPL border router entity in Cooja. The simulations are performed with a randomly generated seed to automate the simulations. The parameters for Contiki for the MAC layer and *Radio Duty Cycle* (RDC) are the default parameters, *csma_driver* and *contikimac_driver* respectively. Due to the length of the messages, we set the parameter UIP_CONF_BUFFER_SIZE to 250 in Contiki OS.

The software used for the proof-of-concept implementations varies depending on the entity in our architecture. For PANA-based experiments, we use OpenPANA 0.2.4 [130] in the Controller side; PANATIKI [175] in the Smart Object side to implement the PANA client and in the Intermediary to implement the PANA Relay.

For the experiments with the LO-CoAP-EAP intermediaries, the Controller implementation of our previous work [67] has been adapted for the intermediaries requirements. Specifically, we use a CoAP library called *cantcoap* [3] that we ported from C++ to C for the proof-of-concept implementation of the LO-CoAP-EAP proxy, relay and stateless proxy. Without loss of generality, we use EAP-PSK [25] to perform the EAP authentication. The

[3] https://github.com/staropram/cantcoap

EAP-PSK keys are 16 bytes long and the EAP identity used in these tests is 14 bytes long. The EAP-PSK implementation, which is common to LO-CoAP-EAP and PANA experiments, is provided by PANATIKI. The AAA server used for the experiments is a RADIUS server, FreeRADIUS version 2.0.2 (freeradius.org).

To carry out a sensible and fair comparison between the different LO-CoAP-EAP intermediaries, we first fix a bootstrapping based on LO-CoAP-EAP *without handshake* and evaluate the three alternatives. Then we change to LO-CoAP-EAP *with handshake* evaluate again. Moreover, we fix PANA relay as the reference standard for our evaluation.

5.4.1 Performance Evaluation

Number and length of messages

The number and length of the messages sent over the network are of great relevance since it influences factors such as the use of the medium (air) and how much time energy are expended to run the protocol. It is expected that the more messages and bytes are sent over the network, more time to complete the bootstrapping and more energy are consumed sending those messages, and well as an increased probability of re-transmission due to packet loss, which also impacts in the bootstrapping and energy consumption and the ability of a solution to finish the bootstrapping before reaching the maximum number of re-transmissions. The message size can also cause fragmentation depending on the radio technology, which also has a negative effect in these parameters due to possible re-transmissions as a consequence of fragment loss.

In Table 5.3, we can see the message size and number of messages associated to each scenario: 1) with the PANA relay as intermediary, 2) the proxy, relay and stateless proxy given that LO-CoAP-EAP *with handshake* is used and 3) using the proxy, stateless proxy and relay as intermediaries given that LO-CoAP-EAP *without handshake* is chosen. In particular, we show the CoAP message size in the communication between the Smart Object and the Intermediary (column *msg*) and between the Intermediary and the Controller, which show the message size as a result of the operation of the intermediaries.

Generally, using an Intermediary only results in an increment in message size in the path between the Intermediary and the Controller compared with a bootstrapping exchange based on LO-CoAP-EAP but not in the number of messages. For example, in the case of LO-CoAP-EAP, the intermediaries do not increment the original number of messages defined in 4. In fact, this number depends exclusively of the bootstrapping: LO-CoAP-EAP with handshake involves 9 messages and LO-CoAP-EAP without handshake involves 7 messages, regardless the usage of the intermediaries or not.

PANA/CoAP-EAP	PANA		LO-CoAP-EAP with handshake				LO-CoAP-EAP without handshake*			
	msg	relay	msg	relay	stateless proxy	proxy	msg	relay	stateless proxy	proxy
PCI / POST (ID)	16	68	29	62	47	47	29 (*s1*)	62 (*s2*)	47 (*s2*)	47(*s2*)
PAR / POST	40	92	6	39	32	33	-	-	-	-
PAN /ACK	40	92	8	41	8	9	-	-	-	-
Req ID / -	48	100	-	-	-	-	-	-	-	-
Rep ID / -	60	112	-	-	-	-	-	-	-	-
PAR/POST(EAP-PSK 1)	56	108	38	71	64	42	36 (*s4*)	69(*s3*)	62 (*s3*)	63 (*s3*)
PAN/ACK(EAP-PSK 2)	84	136	65	98	65	66	69(*s5*)	102(*s6*)	69(*s6*)	70 (*s6*)
PAR/POST(EAP-PSK 3)	84	136	68	101	94	72	68(*s8*)	101 (*s7*)	94 (*s7*)	74(*s7*)
PAN/ACK(EAP-PSK 4)	68	120	48	81	48	49	48 (*s9*)	81(*s10*)	48 (*s10*)	49 (*s10*)
PAR/POST(EAP Success)	88	140	56	89	81	60	56(*s12*)	89(*s11*)	81(*s11*)	62(*s11*)
PAN/ACK	52	104	23	56	23	24	23(*s13*)	56(*s14*)	23(*s14*)	24(*s14*)
TOTAL	636	1028	341	638	462	402	329	560	424	389
TOTAL # messages	11		9				7			

Label *s#* refers to the step number in Fig. 5.2,5.3 and 5.4).
Assumptions:
Proxy-Uri("coap:[aaaa::c30c::3]b]") Uri-Path("b");
Uri-Path("5") (Resource x = 5); *Smart-Object-Info("aaaa::c30c::3")*;
Controller-Info("aaaa::ff:fe00:1"); Proxy-Scheme("coap").

Table 5.3 Message size of the different Intermediaries

With these considerations, the PANA relay is the most taxing solution in terms of number of messages (11) and bytes sent over the networks (1028). The main reason is the PANA design was not initially considered for IoT networks. On the contrary, the LO-CoAP-EAP original design 4reduces the number of messages and the size of each one to carry out an EAP authentication.

Among LO-CoAP-EAP intermediaries, the relay has the biggest increment in message size since its operation is based on encapsulating the original CoAP message inside another. A lesser increment is achieved with the proxies. The proxy and stateless proxy can modify the original messages before forwarding them but do not use encapsulation. If we differentiate between the stateless proxy and the proxy, the latter is more economic in terms of bytes, since it stores per-Smart Object state while the usage of the stateless proxy implies to sent it in some messages (see Chapter 4). [4]

Depending on the type of intermediary, they offer an improvement in term of number of messages with respect to PANA relay, ranging from 19% (when LO-CoAP-EAP *with handshake* is used) to 37% (when LO-CoAP-EAP *without handshake* is used)). Moreover, due to a reduction in the size of the messages, they offer an improvement in term of bytes sent over the network, between 38% (when the relay is used in LO-CoAP-EAP *with handshake*) to 62% (when the proxy is used with LO-CoAP-EAP *without handshake*.

[4] It is worth noting that the stateless proxy has one less byte ACK message in comparison with proxy, since the proxy sets a value for the Token in the ACK but the stateless proxy uses *empty*.

Bootstrapping Time

We now analyze the time it takes to complete the bootstrapping with the assistance of an intermediary. Figure 5.5 shows how the *bootstrapping time* varies with the use of the LO-CoAP-EAP proxy, stateless proxy and relay, assuming first, LO-CoAP-EAP *without handshake* as bootstrapping (Fig. 5.5a, Fig. 5.5b, Fig. 5.5c) and then LO-CoAP-EAP *with handshake* (Fig. 5.5d, Fig. 5.5e, Fig. 5.5f) Moreover, the evaluation is completed comparing these results with a PANA-based bootstrapping using the PANA relay, which is taking as a reference. The median authentication time is shown in Fig. 5.5.

In general, when the length of a message is higher, it costs much time to transmit it. Moreover, the probability of packet loss also increases, which imply re-transmission. It can also cause fragmentation, which increases the time of sending and forwarding the message. The higher number of messages the more time to complete the bootstrapping. When the conditions of the network worsen, that is, when increasing the number of hops between the Controller and the Intermediary (the messages have to be forwarded by more hops, which takes time) and the loss ratio increases between each hop, the probability of packet loss and corresponding re-transmission is higher, which affects negatively to the bootstrapping time. Even more, it is plausible that the maximum number of re-transmission is reached and the bootstrapping is not complete. Thus, there is a direct relation between the message size and the number of messages (see Section 5.4.1) with the bootstrapping time. In short, the fewer and shorter the messages, the lesser time it takes to complete the bootstrapping.

Following these considerations, given for any number of hops and loss ratio and usage of LO-CoAP-EAP bootstrapping *with or without handshake* (Fig. 5.5d, Fig. 5.5e and Fig. 5.5f) the PANA relay takes always the longest bootstrapping time. The reason is clear: the number of messages (11) and length of the messages (the total number of bytes sent over the network are 1028) exceed by far those observed with the LO-CoAP-EAP intermediaries.

Among the LO-CoAP-EAP intermediaries, we first analyze when LO-CoAP-EAP *with handshake* is used (it involves 9 messages). PANA relay is used as reference. Under acceptable network conditions, 0.05 loss ratio and small number of hops (1-3) (Figure 5.5d), there is no substantial difference among intermediaries. In other words, due to the conditions of the network are not severe, the overhead of sending additional bytes over the network does not increase substantially the bootstrapping time (the transmission cost is low thanks to acceptable network conditions and packet loss and corresponding re-transmissions remain low). However the situation changes a little bit when the number of hops increases. Since the usage of the relay involves more bytes per message due to encapsulation than the stateless and the proxy, the worsening of the network makes significantly more costly in terms of time send those packets. On the contrary, stateless proxy and proxy have very similar message

size and upon the same number of messages involved is reduced (7), that re-transmission are similar under these benign network conditions.

When the loss ratio increases to 0.1 (Fig. 5.5e), the relay increases the difference with stateless proxy and proxy, but this two are still resiliance under this network conditions due to the reduced number of messages and similar message size. Only under very rough network conditions, the little differences in message size of the stateless proxy and proxy start to impact due to re-transmissions. That happens when the loss ratio is 0.15 (Fig. 5.5f) number of hops beyond 3, when the stateless proxy poses a higher bootstrapping time than the proxy.

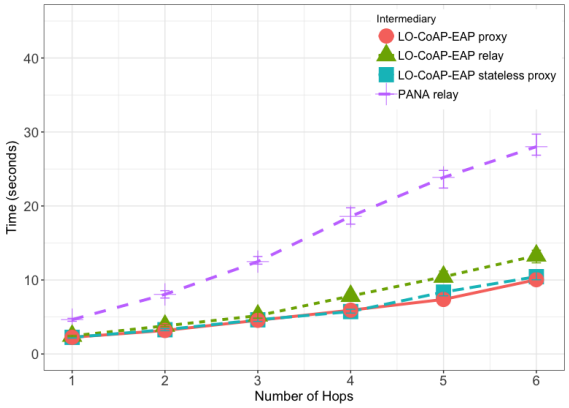
A similar analysis is applicable when the the bootstrapping used is LO-CoAP-EAP *with handshake* (Fig. 5.5d, Fig. 5.5e, Fig. 5.5f). However, the involvement of more messages (9) makes the differences between the intermediates more noticeable even when acceptable network conditions (Fig. 5.5a). The reason is that involving more messages and an increasing number of bytes per message, probability of re-transmissions increases and re-transmitting larger messages under severe network conditions may provoke even further re-transmissions, which increases the bootstrapping time.

In summary, using LO-CoAP-EAP intermediaries, in any variant, offers an improvement ranging from 35% (relay when using LO-CoAP-EAP without handshake) to 76% with respect to exiting standard PANA Relay (the proxy when using LO-CoAP-EAP without handshake). As expected, among the LO-CoAP-EAP intermediaries, the proxy offers a better performance, specially when the network conditions worsen, but followed closely by the stateless proxy, since they have similar message size. The improvements compared to the relay are clear, up to 32% (relay is used with LO-CoAP-EAP *with handshake*).

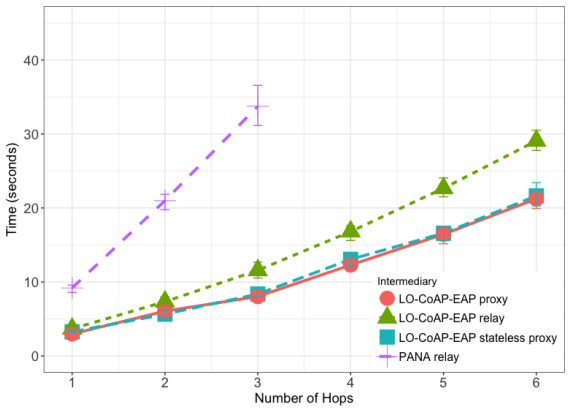
Bootstrapping Success ratio

The success ratio is the relation between the number of completed and initiated bootstrapping. This gives us a way of measuring which solution is more likely to complete the bootstrapping. The success ratio is shown in Fig. 5.6.

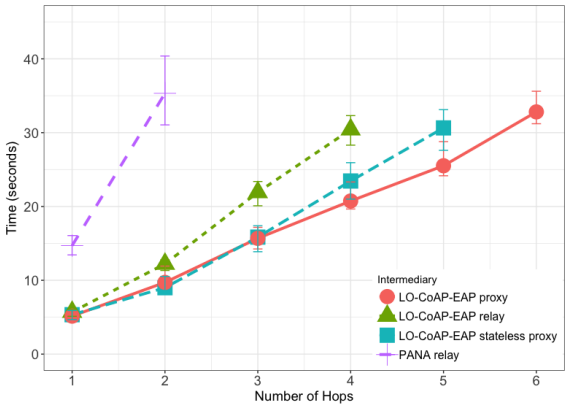
Regardless whether that the LO-CoAP-EAP intermediary operates when the bootstrapping is performed with LO-CoAP-EAP *with or without handshake* at any loss ratio (0.05, 0.1 or 0.15) and number of hops, we can see that the PANA relay is less likely to complete an authentication than any LO-CoAP-EAP intermediary. Again, the reason is that PANA relay involves more messages and each message is longer than any LO-CoAP-EAP intermediary. Having a longer message implies that fragmentation may happen. If a fragment is affected by the loss ratio the probability that the entire packet being re-transmitted increases, so that the maximum number of re-transmissions may be reached before completing the authentication.



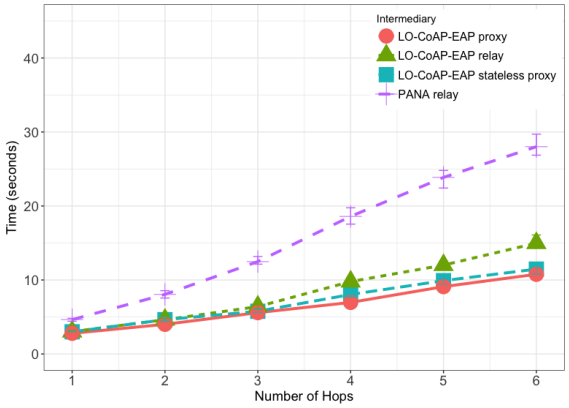
(a) LO-CoAP-EAP *without handshake*, 0.05 loss ratio



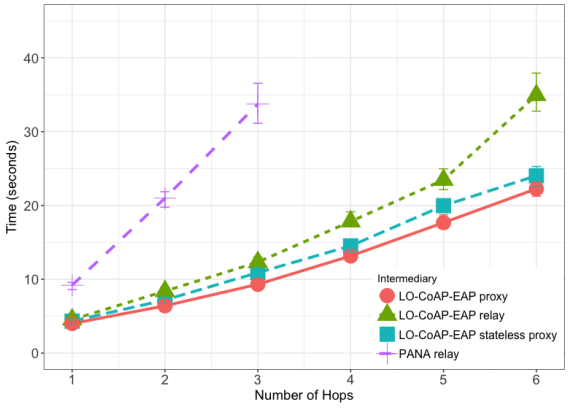
(b) LO-CoAP-EAP *without handshake*, 0.1 loss ratio



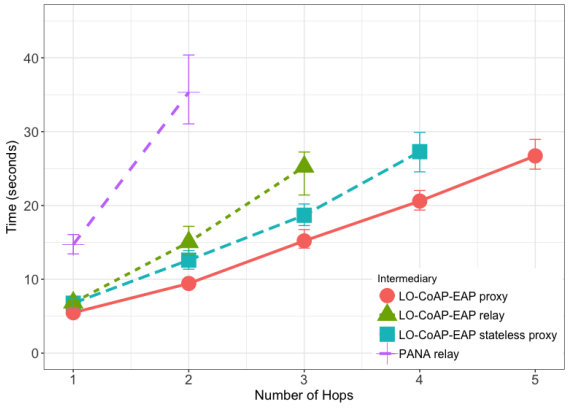
(c) LO-CoAP-EAP *without handshake*, 0.15 loss ratio



(d) LO-CoAP-EAP *with handshake* 0.05 loss ratio



(e) LO-CoAP-EAP *with handshake* 0.1 loss ratio



(f) LO-CoAP-EAP *with handshake* 0.15 loss ratio

Fig. 5.5 Bootstrapping time for LO-CoAP-EAP and PANA with intermediary. Bootstrapping with LO-CoAP-EAP *without handshake* ((**a**), (**b**), (**c**)); Bootstrapping *with handshake* ((**d**), (**e**), (**f**))

In fact, the PANA relay is not able to complete an authentication with an acceptable rate beyond 3 hops..

For LO-CoAP-EAP intermediaries, Fig. 5.6a, Fig. 5.6b and Fig. 5.6c show the comparison when the bootstrapping is performed with LO-CoAP-EAP *without handshake*. PANA relay is used as reference. When network condition are not severe (0.05 loss ratio), Fig. 5.6a, all LO-CoAP-EAP intermediaries how a very similar probability of finishing the bootstrapping since they involve the same number of messages (7) and packet loss is so reduced the re-transmission probability is low. However, when increasing loss ratio (0.1) Fig. 5.6b, the difference between solutions starts to be appreciated with the increment of number of hops, which increases the probability of re-transmissions due to packet loss. In particular, the relay shows a degradation higher than those shown by stateless proxy and proxy. Again, the reason is that the relay is sending larger packets over the network, and due to the increment of loss ratio and number of hops, the probability of re-transmission is higher so it is the probability of reaching the maximum number of re-transmission (4 in the standard CoAP). This deterioration also happens with the stateless proxy and proxy but, since they have similar messages length, the re-transmission is similar to both and they have similar results. The differences between the three intermediaries are more patent when the network conditions worsen a little bit more (0.15 loss ratio Fig. 5.6c). The relay, having larger messages in the communication with the Controller, has more probability of re-transmitting, reaching the maximum number of re-transmission. Moreover, under this conditions, stateless proxy behaves a little bit worse than proxy because, despite the small differences in the message size, they have an impact in the re-transmissions so that the stateless proxy is more likely to reach the maximum number of re-transmissions.

The same analysis and reasoning are applicable when using the LO-CoAP-EAP *with handshake* (Fig. 5.6d, Fig. 5.6e, Fig. 5.6f). In fact they follow the same trend as observed when using the LO-CoAP-EAP *without handshake*. However, the involvement of more messages (9) over the network (which implies more bytes sent according to Table 5.3) and an increasing number of bytes per message, probability of re-transmissions increases and re-transmitting larger messages may provoke further re-transmissions. This is verified because the differences between relay, stateless proxy and proxy are clearer even when low loss ratio (0.05).

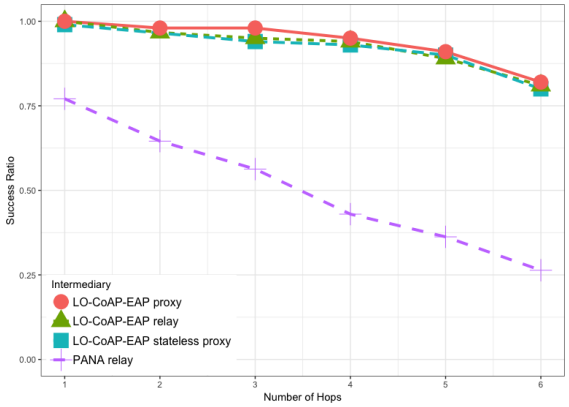
In summary, depending on the scenario, LO-CoAP-EAP intermediaries shows an improvement ranging from 20% (relay when using LO-CoAP-EAP without handshake) up to 68% (proxy when using LO-CoAP-EAP without handshake) over the PANA relay. Comparing between LO-CoAP-EAP alternatives, using proxy or stateless proxy as intermediaries give better results than the relay, though the network conditions worsen proxy provides a

better bootstrapping success ratio due to shorter message size assuming the same number of messages.

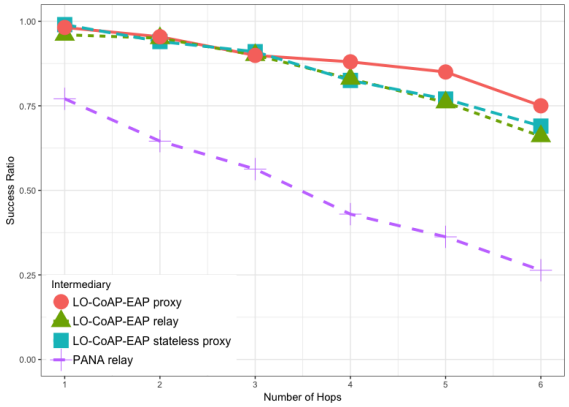
Energy Consumption

We are now interested in knowing how much energy an intermediary uses to assist a Smart Object to join the network. To measure the energy consumption we have used the *Powertrace* tool in Cooja. We show the median energy consumed per each bootstrapping (mJ/bootstrapping) by the intermediary in each scenario. Figure 5.7 shows the total energy consumption of each alternative. It is important to mention that receiving (RX) and, especially, transmitting (TX) data, are the most consuming energy operations due to the radio interface have to be active. Moreover, the longer packet the longer the TX/RX is active and the more packets the more often the network interface is active. According to this, PANA relay is the most energy consuming alternative due to involve more messages (*11*) and their messages are longer than those observed in the LO-CoAP-EAP intermediaries.

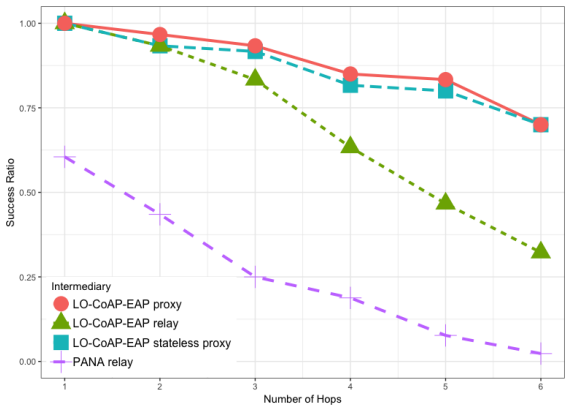
When LO-CoAP-EAP *without handshake* is used as bootstrapping (Fig. 5.7a, Fig. 5.7b, Fig. 5.7c), the LO-CoAP-EAP intermediaries does not show important differences with small number of hops. However, increasing the number of hops and when the loss ratio increases, the energy consumption is higher because the network conditions worsen and more probability of re-transmissions happen in the scenario, with the corresponding increment in the energy consumption to send the messages. What we can observe for any loss ratio is that, proxy and stateless proxy remain in similar values. The reason is the following. Both intermediaries have to send messages to the Smart Object but the number of messages and the size of them are equal because, as seen in Table 5.3 (column *msg*), they send the same messages in the path between the Intermediary and the Smart Object as it has been explained in Chapter 4. Also the intermediaries receive the same number of ACKs and the quantity of bytes from the Smart Object (Table 5.3-column *msg*). When the intermediary sends the ACKs to the Controller, the stateless proxy forwards the ACKs but the message size after the operation is practically the same as the proxy (Table 5.3-column *LO-CoAP-EAP without handshake*/stateless proxy and proxy). The only small differences between both happens in the size of the messages received from the Controller (messages POST). However these small differences make both alternatives statistically indistinguishable in terms of energy consumption. A similar analysis is applicable to the relay. However the relay expends more energy not only in the reception of the messages coming from the Controller (POST) but also in sending the ACKs, because they are both encapsulated in another CoAP message, which makes these messages larger to those observed in the stateless proxy and the proxy. Consequently, the energy consumption increases.



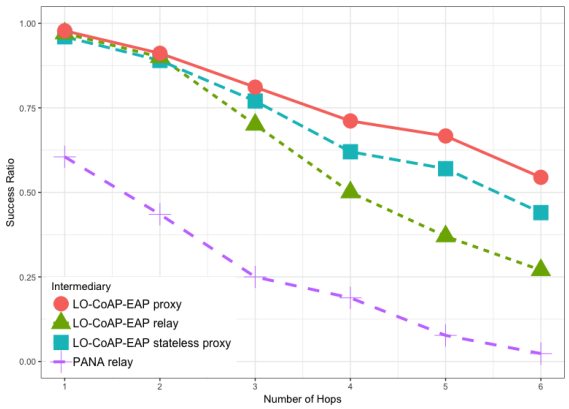
(a) LO-CoAP-EAP *without handshake* 0.05 loss ratio



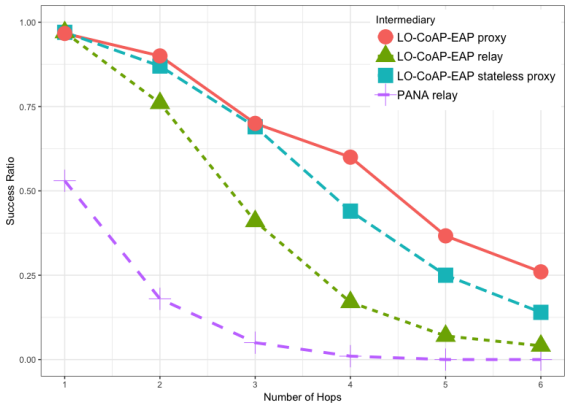
(d) LO-CoAP-EAP *without handshake* 0.05 loss ratio



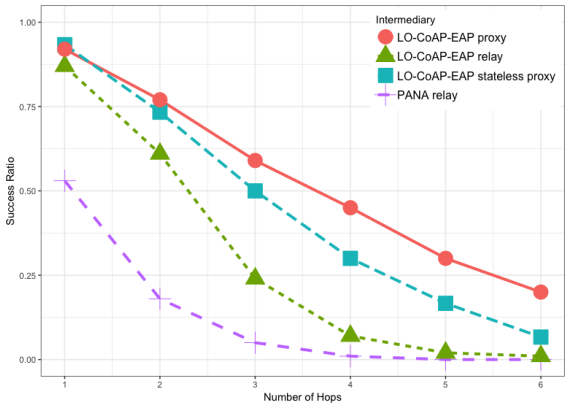
(b) LO-CoAP-EAP *without handshake* 0.1 loss ratio



(e) LO-CoAP-EAP *without handshake* 0.1 loss ratio



(c) LO-CoAP-EAP *without handshake* 0.15 loss ratio



(f) LO-CoAP-EAP *without handshake* 0.15 loss ratio

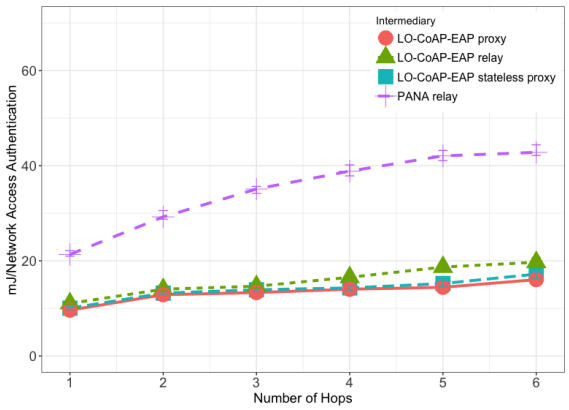
Fig. 5.6 Success ratio for LO-CoAP-EAP and PANA with intermediaries. Boostrapping with LO-CoAP-EAP *with handshake* ((**a**), (**b**), (**c**)); LO-CoAP-EAP Auth. *without handshake* ((**d**), (**e**), (**f**))

A similar trend happens when we change to the bootstrapping based on LO-CoAP-EAP *with handshake* (Fig. 5.7d, Fig. 5.7e, Fig. 5.7f) (9 messages) for exactly the same reasons. As evident, using a different bootstrapping model with more messages makes the energy consumption higher in the three alternatives. Nevertheless, the stateless proxy consumes similarly to proxy and the relay behaves the worst as a consequence of having longer message size, as analyzed when LO-CoAP-EAP *without handshake* is used as bootstrapping. In summary, any LO-CoAP-EAP intermediary offers an improvement ranging from 37% (relay when using LO-CoAP-EAP with handshake) to 67% (proxy when using LO-CoAP-EAP without handshake) compared to the PANA relay. The two best LO-CoAP-EAP alternatives are the proxy and stateless proxy, with very similar energy consumption in every scenario, making the two of them good options to be used as intermediaries.

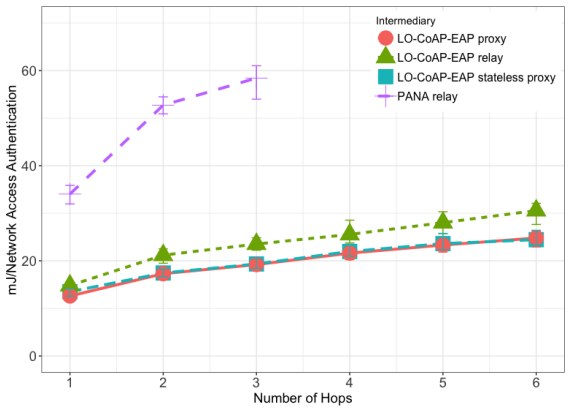
Choosing an alternative would depend on the requirements of the deployment. If we look for the simplest alternative, and the restrictions on the network and energy consumption are not very imposing, the relay fits these requirements. If the network is very restricted, and we need to send the least bytes possible, the proxy is the most efficient, at the cost of the added complexity of managing a state. If we want a trade-off between efficiency and simplicity, we can opt for the stateless proxy. It offers an acceptable performance as well as a version of the proxy that does not store state. In any case, a constrained device with suitable capabilities could deploy a LO-CoAP-EAP proxy instead of a stateless proxy in order to save bytes and energy at the cost of keeping some state. The important part is the Smart Object joining the network will not distinguish what type of Intermediary is interacting with. The downside is the Controller will have to implement the functionality for interacting with any of the possible types of Intermediaries. However, we can assume that the Controller is not so constrained, and it will have no problem to achieve this task.

5.5 Conclusions

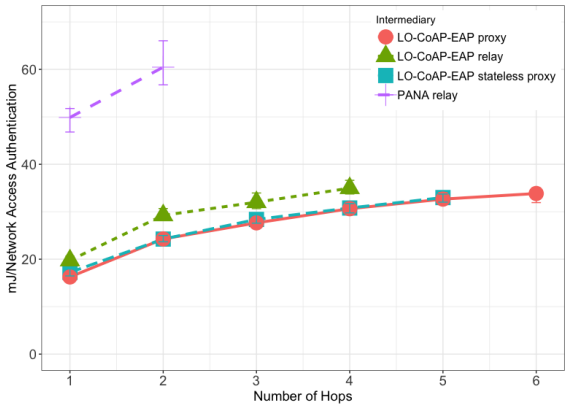
In this chapter, we have extended the LO-CoAP-EAP architecture to include an entity to assist a newcomer smart object in the bootstrapping process in multi-hop networks. We have explored three alternatives: a LO-CoAP-EAP proxy, based on a CoAP proxy as specified in the standard. A LO-CoAP-EAP relay, which is a new entity we have defined that, as opposed to a CoAP proxy, does not store any per-state or alter the messages it receives. Lastly, the LO-CoAP-EAP stateless proxy, a new entity designed as a trade-off between the two previous Intermediaries as it does not keep any per-Smart Object state, unlike the LO-CoAP-EAP proxy, at the cost of sending the state over the network.



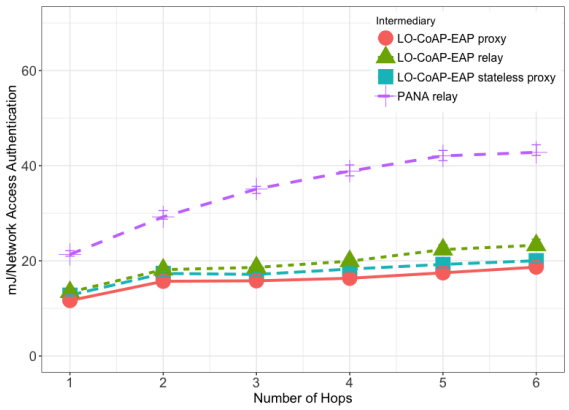
(a) LO-CoAP-EAP *without handshake* 0.05 loss ratio



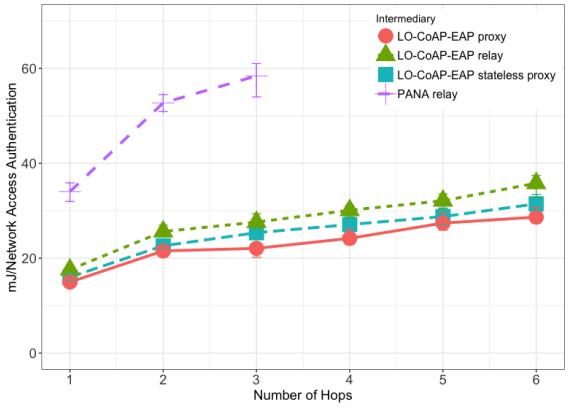
(b) LO-CoAP-EAP *without handshake* 0.1 loss ratio



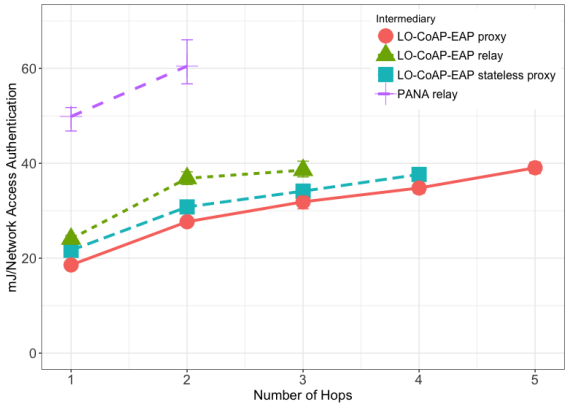
(c) LO-CoAP-EAP *without handshake* 0.15 loss ratio



(d) LO-CoAP-EAP *with handshake* 0.05 loss ratio



(e) LO-CoAP-EAP *with handshake* 0.1 loss ratio



(f) LO-CoAP-EAP *with handshake* 0.15 loss ratio

Fig. 5.7 Total energy consumption for LO-CoAP-EAP and PANA with intermediaries. Bootstrapping with LO-CoAP-EAP *without handshake* ((**a**), (**b**), (**c**)); LO-CoAP-EAP *with handshake* ((**d**), (**e**), (**f**))

Based on our experiments, we can conclude that using CoAP-based entities to act as intermediaries, shows substantial improvements in the three alternatives with respect to the PANA relay, thanks to having a shorter message size and involving fewer messages. Moreover, the LO-CoAP-EAP stateless proxy represents a better trade-off than the LO-CoAP-EAP relay and proxy. Nevertheless, we consider that since each alternative has its advantages, any of them could be deployed depending on the capabilities of the Smart Objects that implement the intermediary. This would not represent any problem to the Smart Object performing the bootstrapping, because the exchange is always the same for the Smart Object. The Controller is the entity that has to implement the logic to support each alternative, but we understand that this is not really an issue since the Controller is assumed to be a not very constrained entity.

Chapter 6

Conclusions and future work

In this chapter we summarize the work done in this thesis and its main contributions. Finally, we discuss future work and indicate possible future directions.

6.1 Summary and main contributions

The IoT is a fairly recent concept that leads the trend of connecting smart objects to the Internet. Smart objects are devices that interact with their environment by reporting about what they sense or performing some action, depending if they are sensors or actuators. This basic functionality is expanded into services running on top of these devices that can be leveraged to easily monitor and manage different appliances in homes, buildings, enterprises, cities, etc. The IoT is of great interest not only in research, but also, because of its potential, in standardization organizations and companies such as IEEE, IETF, IPSO, Zigbee, OMA.

With the goal of providing most smart objects with connectivity to the Internet there are efforts in progress like the adaptation of IPv6 to different link-layer technologies. This has been done in 6LoWPAN and IEEE 802.15.4, also known as LR-WPAN [82], and very recently with Bluetooth [139]. Currently it is being developed for a new set of technologies called LPWAN [59], achieving a homogenized way of providing Internet connectivity.

Putting into practive the Internet of Things has its challenges [149]. Among these, security stands out because of the challenges added by the IoT to those already present in the Internet today that arise from the vast amount of devices expected to be part of the Internet of Things. To secure the Internet of Things, there is agreement in various works that certain security services have to be present [123, 170, 81, 87, 100, 109, 215] such as authentication (identifying the smart object), authorization (what permissions this smart object has), key management (distributing or deriving the necessary key material to protect the communications, etc. These security processes are used for different purposes: to securely

interact with other devices, to request credentials, access services, etc. and also, to be part of what is known as bootstrapping. Bootstrapping provides the basis for the secure integration and operation of a smart object in a network. It entails the authentication, authorization to be part of the security domain, the provision of the necessary key material and auditing of the use of the network resources by the smart object.

In this dissertation, we look at the state of the art in bootstrapping in the IoT and we find that existing solutions focus on deployments with a simple management of credentials, using, generally, pre-configured key material, and by simply running a security association protocol to protect the communications between the smart object and the Controller of the domain. This approach is not feasible in extensive deployments, where several administrative domains are part of a deployment, such as a university with several campuses, and one or more buildings per campus, etc. and with the possibility of having devices from other organizations. We call these large-scale deployments. Furthermore, current solutions do not account for the set of requisites that we elicited for large scale IoT networks. A bootstrapping solution for use in large-scale IoT deployments needs to provide: 1) Lightweight protocols, due to the diversity of devices and radio technologies with different constraints; 2) Link-layer independence, to support different radio technologies that become part of the IoT, to ease the management and to provide interoperability; 3) Flexible authentication, because of the different capabilities in the hardware of the smart objects or other restrictions in the authentication method; 4) Flexible key management, to suit the different technologies or standard used to secure the communications; 5) Robust and flexible authorization, to provide well established policies focused on bootstrapping and network access; 6) Identity federation, to support co-existence of devices from different organizations, 7) Accounting to track the use of the resources by the smart objects; 8) Support multi-hop topologies; 9) Reuse code whenever possible, due to memory constraints of some devices; 10) Adaptation to the constraints of LPWAN networks.

With these requirements in mind, this thesis designs a bootstrapping service for the Internet of Things for large-scale deployments. We review different technologies to design a bootstrapping service for the Internet of Things that covers the aforementioned requisites and we build the service based on three pillars: AAA infrastructures, EAP and CoAP. AAA infrastructures have been used for decades to provide federated network access services. They are robust, reliable, support identity federation and are used in conjunction with EAP, which provides the flexibility to choose the authentication method as well as key management to protect the communications. To cover the requisites of a lightweight solution as well as link layer independence, we have designed a bootstrapping service for the Internet of Things, using the Constrained Application Protocol CoAP. The service entails the design of a new

EAP lower layer, which is a protocol to transfer EAP messages between the EAP peer and the EAP authenticator, which is the constrained link in the IoT, using CoAP as transport. The use of CoAP was key to the design, as it is an application level protocol that runs on top of UDP, which makes it independent of the link-layer; it is designed for constrained devices and networks, with low overhead and parsing complexity. For these reasons CoAP allows us to design a new EAP lower layer that not only reduces the overhead of the lower layer in comparison with the current standard EAP lower layer for the IoT (PANA), but also provides the operational homogeneity of having the bootstrapping service as any other CoAP service, so avoiding having to use an additional library for the sole purpose of bootstrapping. Next we summarize the three main contributions of this dissertation.

In *Chapter 3*, we design the lightweight bootstrapping service for large-scale IoT networks, called CoAP-EAP, covering the first objective (O1) and, partially, the last objective (O4). We describe the architecture, the general flow of operation and design a new EAP lower layer using CoAP that is part of the bootstrapping service. After the bootstrapping has been completed, it provides the necessary key material to run the security association protocol that best fits the IoT system. For instance, in IEEE 802.15.4 the smart object could run DTLS with the Controller, using generated key material derived from the MSK, as a consequence of the EAP authentication. CoAP-EAP has been tested in IEEE 802.15.4 networks using the simulations with the Cooja simulator (a Contiki O.S. network simulator), and we have compared it with PANA, the current bootstrapping solution for IoT. CoAP-EAP outperforms in every measure PANA, bootstrapping time, percentage of successful bootstrapping processes and energy consumption. The results show a reduction of $\approx 50\%$ the number of bytes used by the lower layer, a reduction in bootstrapping time of up to $\approx 56\%$ and a reduction in energy consumption up to 46%.

After the initial design, we were contacted by IMT Atlantic (former Telecom Bretagne) due to their interest in CoAP-EAP and its applicability in LPWAN. LPWAN, which is fairly new in IoT, encompasses different technologies that allow large distance communications with very low energy consumption and very low bandwidth. Each technology designs their own protocols to secure the communications, mostly using pre-shared keys, or running a security association protocol to join the network, as in the case of LoRaWAN. Therefore, in *Chapter 4*, due to the high constraints in the link of LPWAN, we designed an optimized version of CoAP-EAP that considers the limitations of LPWAN, which called for a redesign and an optimization of the protocol to reduce the overhead, sending only the necessary information to complete the bootstrapping. We call it Low Overhead CoAP-EAP (LO-CoAP-EAP). We eliminated the messages that were optional, sending fewer bytes over the network, as well as simplifying some of the data representation that was susceptible to this. LO-CoAP-EAP,

has been tested in a real LoRa deployment in Rennes, France, outperforming the original design, and consequently, PANA, showing that for the initial process of bootstrapping with federation support is feasible in these very constrained environments, with a trade-off between performance and interoperability. To verify the improvement in the design, we also tested LO-CoAP-EAP in IEEE 802.15.4 in the Cooja simulator, and compared the performance to CoAP-EAP and PANA and verified that indeed provides the bootstrapping service with improved performance in all the parameters tested —time, percentage of successful bootstrapping processes and energy. We obtain a reduction up to 72% in the message size overhead in comparison with PANA and a 38% reduction in comparison with CoAP-EAP. The contribution of Chapter 4 accomplishes the second objective (O2) and, partially, the last objective (O4).

Chapter 5, considers the scenario of a multi-hop network where the smart objects can not reach the Controller autonomously, so covering the third objective (O3) and completing the last objective (O4). When the device is not able to reach the Controller through a series of hops, we need to provide a way of assisting the smart object in the bootstrapping process, for which we design three CoAP-based intermediary entities to be used in the context of bootstrapping, and used, in this case for our bootstrapping service. We use the CoAP proxy intermediary entity as basis for the proxy intermediary. This entity is able to store a state related to the exchange and analyze the messages to make decisions and modify them. We also design a CoAP relay, which is not defined in the CoAP standard. The design of the CoAP relay is motivated by the search of a stateless alternative that does not have to analyze the messages, so minimizing the processing of these. It simply creates a CoAP tunnel to send the message from the smart object to the Controller, which in turn increases the message size. Finally, we design a stateless proxy, a hybrid that selects characteristics from the previous two. The stateless proxy does not saves a state related to the authentication (this information is sent in the messages) acting as proxy, with the ability to analyze the messages and modify them, so reducing the increase in message size of the CoAP relay. These designs leave the bootstrapping exchange unmodified to the client, who does not know until the last exchange that it was talking with the Controller through an intermediary entity. We evaluate each intermediary, using the Cooja simulator, and compare it to the PANA relay. We evaluate the overhead the intermediaries add to the original message sent to (and by) the smart object, the bootstrapping time and the energy consumption and the percentage of successful bootstrappings of each solution. We find an improvement ranging from 35% to 76% over the existing standard PANA relay in bootstrapping time, and an energy consumption improvement ranging from 37% to 67%.

6.2 Future work

As we we have described, this dissertation answers the set of objectives identified at the onset. However, in the context of CoAP-EAP there is work that can be done to further improve the bootstrapping service. Concretely, we identify different points that are subject to further improvement of the existing solutions:

- Test LO-CoAP-EAP with other technologies such as Bluetooth. Bluetooth and it recent adaptation of IPv6 and the work towards the support mesh networks makes it an interesting candidate to test CoAP-EAP further with other radio technologies.
- Validate the protocol in real world deployments in the context of the industrial Doctorate, to confirm the findings in this thesis in the context of industrial applications.
- Test different security associations as a consequence of the bootstrapping process and evaluate the performance and suitability of different security association protocols in different deployments and with different technologies.
- Collaborate in standardization organizations like the IETF in areas such as LPWAN to promote the present work and collaborate in related issues.

Throughout this dissertation, future work is identified can be done related to the bootstrapping service and new lines of research can be opened up in the area of communication security in IoT. Some of these are:

- Evaluate the use compression techniques that complement the approach of LO-CoAP-EAP; to reduce the overhead, sending fewer bytes over the network. As an example of compression techniques there is one being developed in the context of LPWAN, called LPWAN *Static Context Header Compression* (SCHC). It is being applied, to CoAP [128, 177] among other protocols. These techniques can be leveraged to further improve the bootstrapping process in these kinds of networks.
- New work related to the re-use of existing protocols to secure the communications as part of the post-bootstrapping phase. In specific case of DTLS, optimizing it to the constraints of IoT is being discussed in different documents of the IETF [182, 27, 63] of which we are collaborating in a specific proposal [64] to reuse the DTLS record and provide advance features such as object security, without having to resort to new protocols. Recently a mailing list named *Application Transport LAyer Security* (ATLAS) has been created to discuss the work around the reuse of DTLS in constrained scenarios.

- Explore methods of fast re-authentication in the context of CoAP-EAP. Once the bootstrapping is completed, the state of a current bootstrapping can be used to speed up a new bootstrapping process to renew it. Another use case is when a smart object changes Controller. This opens the possibility of studying the different approaches to make the process of changing the Controller (known as *mobility*) more efficient and faster than through a normal bootstrapping, whenever possible.

Glossary

3GPP *3rd Generation Partnership Project*. 16, 51, 52, 60

6LoWPAN *IPv6 over Low power Wireless Personal Area Networks*. 3, 28, 29, 50, 55, 80, 94, 95, 147

6TiSCH *IPv6 over the TSCH mode of IEEE 802.15.4e*. xxvii, 6, 47–50, 58, 125, 126, 128, 134

AAA *Authentication Authorization Accounting*. xvii–xix, xxvii, 16–24, 30–36, 38–40, 43, 52, 53, 56–63, 65, 67, 71, 73–76, 78, 91, 94–100, 105, 107, 108, 117, 121, 125, 136, 148

ACE *Authentication and Authorization for Constrained Environments*. 6, 9, 50, 51, 58, 74

ACK *Acknowledgment*. 37, 38, 64, 65, 68, 71, 72, 76, 79, 106, 119, 126, 131, 134, 137, 142

AEAD *Authenticated Encryption with Associated Data*. 47

AES *Advanced Encryption Standard*. 75, 97, 102

AES-CCM *AES in Counter with CBC-MAC (CCM) Mode*. 75

AKA *Authentication and Key Agreement*. 8

ANIMA *Autonomic Networking Integrated Model and Approach*. 16, 51

API *Application Program Interface*. 39

ASCII *American Standard Code for Information Interchange*. 69, 70, 102

ASRK *Application-Specific Root Key*. 67, 71, 79

ATLAS *Application Transport LAyer Security*. 151

AVP *Attribute-Value Pair*. 79

BLE *Bluetooth Low Energy*. 29

BOOTP *Bootstrap Protocol*. 7

BRSKI *Bootstrapping Remote Secure Key Infrastructures*. 16, 50

CBOR *Concise Binary Object Representation*. 46, 50, 74

CNP *Configuration Network Protocol*. 39

CoAP *Constrained Application Protocol*. xvii–xix, xxvii, 3, 14, 16, 21–24, 28, 36–38, 45, 47–50, 52, 53, 57–64, 67–70, 72–75, 78–81, 84–86, 91, 94–96, 98–100, 105–107, 110, 111, 114, 116, 117, 121, 123–126, 128, 129, 131, 133, 135–137, 141, 142, 144, 148–151

CoRE *Constrained RESTful Environments*. 3, 6, 22, 50, 73, 78, 99, 111

COSE *CBOR Object Signing and Encryption*. 46, 47, 51, 69

CPU *Central Processing Unit*. xv, 4, 12, 37, 78, 86, 111, 114, 121

CSMA *Carrier Sense Multiple Access*. 78

CWT *CBOR Web Token*. 50

DASH *Developers' Alliance for Standards Harmonization of ISO 18000-7*. 29, 94, 96

DHCP *Dynamic Host Configuration Protocol*. 7

DoS *Denial of Service*. 108

DTLS *Datagram Transport Layer Security*. 12, 14–17, 23, 28, 45–47, 49–52, 55, 59–61, 67–70, 73, 75–77, 126, 127, 134, 149, 151

EAP *Extensible Authentication Protocol*. xvii–xix, xxvii, 7, 15–21, 23, 24, 32–36, 38–43, 52, 57–63, 65, 67–81, 84, 85, 91, 94–96, 98–103, 105, 107, 109, 110, 116, 119–121, 123–126, 128–130, 135–137, 148, 149

EAP-KMF *EAP Key Management Framework*. 53, 62

EAPoL *EAP over LAN*. 15, 20, 40

EDHOC *Ephemeral Diffie-Hellman Over COSE* . 46

EMSK *Extended Master Session Key*. 35, 36

EP *Enforcement Point*. 38–40

ESP *Encapsulating Security Payload*. 109

EST *Enrollment over Secure Transport*. 16, 48–50

ETSI *European Telecommunications Standards Institute*. 50, 58, 59

GBA *Generic Bootstrapping Architecture*. 52, 60

HIP *Host Identity Protocol*. 15

HIP-DEX *Host Identity Protocol Diet EXchange*. 12, 14, 15, 45, 58, 60

HIT *Host Identity Tag*. 44

HTTP *Hypertext Transfer Protocol*. 36, 49, 60, 117

IdP *Identity Provider*. 10, 11, 13, 22, 31, 33, 63, 65, 73–75

IEEE *Institute of Electrical and Electronics Engineers*. 6, 15, 25, 40, 50, 58, 147

IETF *Internet Engineering Task Force*. 3, 6, 9, 16, 30, 32, 50, 58, 74, 80, 99, 124, 125, 147, 151

IKE *Internet key exchange*. 42, 43

IKEv2 *Internet key exchange version 2*. xxvii, 12, 15, 38, 42, 43, 59

IoT *Internet of Things*. xv–xix, xxix, 1–9, 12–14, 16, 17, 19–25, 27–30, 36–38, 41, 42, 44, 46, 49–53, 55–58, 60–62, 68, 74, 76, 80, 84, 91, 93–98, 108, 116, 121, 123, 124, 137, 147–149, 151

IPSO *Internet Protocol for Smart Objects*. 6, 50, 52, 58, 60, 147

ISM *Industrial, Scientific and Medical*. 29, 94

JP *Joining Proxy*. 48, 125

JRC *Join Registrar/Coordinator*. 48, 125, 126

JSON *JavaScript Object Notation*. 74

KDF *Key Derivation Function*. 47, 69, 75, 76, 84, 102

KMF *Key Management Framework*. 20, 34–36, 38, 41, 65, 67, 73, 99, 101

KMP *Key Management Protocol*. 15, 42–44, 51, 59

LAN *Local Area Network*. 33, 40

LED *Light-Emitting Diode*. 4

LoRa *Long Range*. xvii, xviii, xxix, 23, 26, 29, 95, 97, 106–108, 116–121, 150

LPWAN *Low-Power Wide-Area Network*. xvii–xix, 3, 12, 16, 17, 23, 24, 26, 28–30, 49, 51, 68, 91, 93–101, 106–110, 114, 116, 119, 121, 147–149, 151

LR-WPAN *Low-Rate Wireless Personal Area Network*. xvii–xix, 3, 12, 14, 23, 24, 29, 55, 93–95, 106, 108, 121, 147

LWM2M *Lightweight M2M*. 52, 60

MAC *Message Authentication Code*. 2, 15, 35, 51, 69, 75, 78, 80, 106, 107, 135

MID *Message ID*. 130–132, 134

MIP6 *Mobile IPv6*. 7

MIT *Massachusetts Institute of Technology*. 1

MSK *Master Session Key*. 35, 36, 40, 67, 70, 73, 100, 101, 129, 130, 149

MTU *Maximum Transmission Unit*. 29, 62, 79, 80

NAI *Network Access Identifier*. 100

NAS *Network Access Server*. 31, 32

OIDC *OpenID Connect*. 18, 19

OMA *Open Mobile Alliance*. 6, 50, 52, 58, 147

OSCORE *Object Security for Constrained RESTful Environments*. 12, 45, 47, 50, 51, 126, 127, 130

OTA *Over The Air*. 102

PAA *PANA Agent*. 38–40, 59, 78

PaC *PANA Client*. 38–40, 59, 125

PAN *PANA Answer*. 39, 79

PANA *Protocol for Carrying Authentication for Network Access*. xvii–xix, xxviii, xxix, 15–17, 20, 21, 23, 24, 26, 38–40, 43, 51, 52, 55, 58, 59, 76–79, 81, 84, 96, 108, 109, 111, 121, 123, 124, 126, 135–146, 150

PAR *PANA Request*. 39, 79

PCI *PANA-Client-Initiator*. 79

PEMK *PaC-EP Master Key*. 40

PNAC *Port-based Network Access Control*. 40

PRE *PANA relay*. 38, 39, 125

PRF *Pseudo Random Function*. 102

PSK *Pre-Shared Key*. 8, 17, 20, 45, 46, 71, 96, 126, 128

RADIUS *Remote Authentication Dial-In User Service*. 31, 32, 65, 74, 99, 100, 119, 136

RDC *Radio Duty Cycling*. 78, 110

RPK *Raw Public Key*. 45, 46

RPL *IPv6 Routing Protocol for LLN*. 110, 135

RST *Reset*. 37

SAML *Security Assertion Markup Language*. 18, 74

SAP *Security Association Protocol*. 36, 53, 134

SCHC *Static Context Header Compression*. 151

SIG *Special Interest Group*. 29

SNMP *Simple Network Management Protocol*. 39

TCP *Transmission Control Protocol*. 32, 45

TDF *Telediffusion de France*. 116

TEK *Transient EAP Keys*. 36

TELCO *Telephone Company*. 52

TLS *Transport Layer Security*. 8, 16, 32, 45, 49, 60

TSK *Transient Session Keys*. 36, 102

UDP *User Datagram Protocol* . 15, 21, 28, 32, 37, 38, 42, 43, 45, 62, 117, 129, 130, 132, 133, 149

URI *Uniform Resource Identifier*. 50, 64, 68, 70, 71, 73, 99, 103

W3C *World Wide Web Consortium*. 50, 52, 58, 60

WAN *Wide Area Network*. 105

WG *Working Group*. 50, 51, 58, 74, 80, 94, 99, 124, 125

WI-SUN *Wireless Smart Ubiquitous Network*. 94

WIFI *Wireless Fidelity*. 20

WiMAX *Worldwide Interoperability for Microwave Access*. 94, 102

WLAN *Wireless Local Area Network*. 40

WPA *Wi-Fi Protected Access*. 20, 41

WPAN *Wireless Personal Area Network*. xvii

WSN *Wireless Sensor Network*. 23

References

[1] (2010). IEEE Standard for Local and metropolitan area networks–Port-Based Network Access Control. *IEEE Std 802.1X-2010 (Revision of IEEE Std 802.1X-2004)*, pages 1–205.

[2] (2011). Smart Device Communications Reference Architecture - TR50_ETSI-20110321-002.

[3] (2014). Using cooja test scripts to automate simulations. Web blog post, github.com.

[4] (2015). Web of things security. Web blog post, github.com. https://github.com/w3c/web-of-things-framework/blob/master/security.md.

[5] (2016a). Ieee recommended practice for transport of key management protocol (kmp) datagrams. *IEEE Std 802.15.9-2016*, pages 1–74.

[6] (2016b). 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. *IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012)*, pages 1–3534.

[7] (2016c). Ieee standard for low-rate wireless networks. *IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011)*, pages 1–709.

[8] (2017). IEEE Standard for Local and metropolitan area networks–Media Access Control (MAC) Security - Amendment 3:Ethernet Data Encryption devices. *IEEE Std 802.1AEcg-2017 (Amendment to IEEE Std 802.1AE-2006 as amended by IEEE Std 802.1AEbn-2011 and IEEE Std 802.1AEbw-2013)*, pages 1–143.

[9] 5gensure (1995). AAA (Authentication, Authorization and Accounting). Technical report. [Online; accessed 20-May-2018].

[10] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and Levkowetz, H. (2004). Extensible Authentication Protocol (EAP). RFC 3748 (Proposed Standard). Updated by RFCs 5247, 7057.

[11] Aboba, B. and Calhoun, P. (2003). RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP). RFC 3579 (Informational). Updated by RFC 5080.

[12] Aboba, B., Simon, D., and Eronen, P. (2008). Extensible Authentication Protocol (EAP) Key Management Framework. RFC 5247 (Proposed Standard).

[13] Adame, T., Barrachina, S., Bellalta, B., and Bel, A. (2017). Hare: Supporting efficient uplink multi-hop communications in self-organizing lpwans. *arXiv preprint arXiv:1701.04673*.

[14] Adelantado, F., Vilajosana, X., Tuset-Peiro, P., Martinez, B., and Melia, J. (2016). Understanding the limits of LoRaWAN. *arXiv preprint arXiv:1607.08011*.

[15] Adelantado, F., Vilajosana, X., Tuset-Peiro, P., Martinez, B., Melia-Segui, J., and Watteyne, T. (2017). Understanding the Limits of LoRaWAN. *IEEE Communications Magazine*, 55(9):34–40.

[16] Andrews, J. G., Ghosh, A., and Muhamed, R. (2007). *Fundamentals of WiMAX: understanding broadband wireless networking*. Pearson Education.

[17] Arkko, J. and Haverinen, H. (2006). Extensible Authentication Protocol Method for 3rd Generation Authentication and Key Agreement (EAP-AKA). RFC 4187 (Informational). Updated by RFC 5448.

[18] ARM, A. (2009). Security technology building a secure system using trustzone technology (white paper). *ARM Limited*.

[19] Ashton, K. et al. (2009). That 'internet of things' thing. Technical Report 7.

[20] Aura, T. and Sethi, M. (2017). Nimble out-of-band authentication for EAP (EAP-NOOB). Internet-Draft draft-aura-eap-noob-02, Internet Engineering Task Force. Work in Progress.

[21] Barrachina-Muñoz, S. and Bellalta, B. (2017). Learning optimal routing for the uplink in lpwans using similarity-enhanced epsilon-greedy. *arXiv preprint arXiv:1705.08304*.

[22] Barrachina-Muñoz, S., Bellalta, B., Adame, T., and Bel, A. (2017). Multi-hop communication in the uplink for lpwans. *Computer Networks*, 123:153 – 168.

[23] Beecher, P. (2017). Wi-sun alliance-interoperable communications solutions.

[24] Bergmann, O., Gerdes, S., Schäfer, S., Junge, F., and Bormann, C. (2012). Secure bootstrapping of nodes in a CoAP network. In *Wireless Communications and Networking Conference Workshops (WCNCW), 2012 IEEE*, pages 220–225.

[25] Bersani, F. and Tschofenig, H. (2007). The EAP-PSK Protocol: A Pre-Shared Key Extensible Authentication Protocol (EAP) Method. RFC 4764 (Experimental).

[26] Bhattacharyya, A., Bandyopadhyay, S., Pal, A., and Bose, T. (2016). Constrained Application Protocol (CoAP) Option for No Server Response. RFC 7967 (Informational).

[27] Bhattacharyya, A., Bose, T., Ukil, A., Bandyopadhyay, S., and Pal, A. (2018). Lightweight establishment of secure session (less) on coap. Internet-Draft draft-I-D.draft-bhattacharyya-dice-less-on-coap, Internet Engineering Task Force. Work in Progress.

[28] Billet, O., Gilbert, H., and Ech-Chatbi, C. (2005). Cryptanalysis of a white box AES implementation. In *Selected Areas in Cryptography*, pages 227–240. Springer.

[29] Black, D. and McGrew, D. (2008). Using Authenticated Encryption Algorithms with the Encrypted Payload of the Internet Key Exchange version 2 (IKEv2) Protocol. RFC 5282 (Proposed Standard).

[30] Blazquez Rodriguez, A. (2014). Security and aaa architectures in an iot marketplace. Master's thesis.

[31] Bohli, J. M., Skarmeta, A., Moreno, M. V., García, D., and Langendörfer, P. (2015). SMARTIE project: Secure IoT data management for smart cities. In *2015 International Conference on Recent Advances in Internet of Things (RIoT)*, pages 1–6.

[32] Bormann, C., Castellani, A. P., and Shelby, Z. (2012). Coap: An application protocol for billions of tiny internet nodes. *IEEE Internet Computing*, 16(2):62.

[33] Bormann, C., Ersue, M., and Keranen, A. (2014). Terminology for Constrained-Node Networks. RFC 7228 (Informational).

[34] Bormann, C. and Hoffman, P. (2013). Concise Binary Object Representation (CBOR). RFC 7049 (Proposed Standard).

[35] Calhoun, P., Loughney, J., Guttman, E., Zorn, G., and Arkko, J. (2003). Diameter Base Protocol. RFC 3588 (Proposed Standard). Obsoleted by RFC 6733, updated by RFCs 5729, 5719, 6408.

[36] Cirani, S., Ferrari, G., and Veltri, L. (2013). Enforcing security mechanisms in the ip-based internet of things: An algorithmic overview. *Algorithms*, 6(2):197–226.

[37] Clark, A., Huang, R., and Wu, Q. (2013). RTP Control Protocol (RTCP) Extended Report (XR) Block for Burst/Gap Discard Metric Reporting. RFC 7003 (Proposed Standard).

[38] Clausen, T., Herberg, U., and Philipp, M. (2011). A critical evaluation of the ipv6 routing protocol for low power and lossy networks (rpl). In *Wireless and Mobile Computing, Networking and Communications (WiMob), 2011 IEEE 7th International Conference on*, pages 365–372.

[39] Crockford, D. (2006). The application/json Media Type for JavaScript Object Notation (JSON). RFC 4627 (Informational). Obsoleted by RFC 7159.

[40] Croft, W. and Gilmore, J. (1985). Bootstrap Protocol. RFC 951 (Draft Standard). Updated by RFCs 1395, 1497, 1532, 1542, 5494.

[41] Dantu, R., Clothier, G., and Atri, A. (2007). Eap methods for wireless networks. *Computer Standards & Interfaces*, 29(3):289–301.

[42] Das, S. and Ohba, Y. (2012). Provisioning credentials for coap applications using eap. Internet-Draft draft-I-D.draft-ohba-core-eap-based-bootstrapping, Internet Engineering Task Force. Work in Progress.

[43] Das, S. and Ohba, Y. (2012 - work in progress). Provisioning credentials for coap applications using eap. Internet-Draft draft-I-D.draft-ohba-core-eap-based-bootstrapping, IETF Secretariat. https://tools.ietf.org/html/draft-I-D.draft-ohba-core-eap-based-bootstrapping.

[44] de Laat, C., Gross, G., Gommans, L., Vollbrecht, J., and Spence, D. (2000). Generic AAA Architecture. RFC 2903 (Experimental).

[45] Deering, S. and Hinden, R. (1995). Internet Protocol, Version 6 (IPv6) Specification. RFC 1883 (Proposed Standard). Obsoleted by RFC 2460.

[46] DeKok, A. (2015). The Network Access Identifier. RFC 7542 (Proposed Standard).

[47] der Stok, P. V., Kampanakis, P., Kumar, S. S., Richardson, M., Furuhed, M., and Raza, S. (2018). Est over secure coap (est-coaps). Internet-Draft draft-I-D.draft-ietf-ace-coap-est, Internet Engineering Task Force. Work in Progress.

[48] Dierks, T. and Rescorla, E. (2008). The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard). Updated by RFCs 5746, 5878, 6176, 7465, 7507, 7568, 7627, 7685, 7905, 7919.

[49] Droms, R. and Arbaugh, W. (2001). Authentication for DHCP Messages. RFC 3118 (Proposed Standard).

[50] Duffy, P., Chakrabarti, S., Cragie, R., Ohba, Y., and Yegin, A. (2011). Protocol for Carrying Authentication for Network Access (PANA) Relay Element. RFC 6345 (Proposed Standard).

[51] Dunkels, A., Eriksson, J., Finne, N., and Tsiftes, N. (2011). Powertrace: Network-level power profiling for low-power wireless networks. Technical Report T2011:05, Swedish Institute of Computer Science.

[52] Dunkels, A., Gronvall, B., and Voigt, T. (2004). Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*, pages 455–462.

[53] Eronen, P., Hiller, T., and Zorn, G. (2005). Diameter Extensible Authentication Protocol (EAP) Application. RFC 4072 (Proposed Standard). Updated by RFCs 7268, 8044.

[54] Eronen, P. and Tschofenig, H. (2005). Pre-Shared Key Ciphersuites for Transport Layer Security (TLS). RFC 4279 (Proposed Standard).

[55] ETSI (2011). 102 690 v1. 1.1 (2011-10):machine-to-machine communications (m2m); functional architecture. *ETSI Technical Specification*.

[56] evolved intelligence.com (2017). There's No Time Like the Present to Secure 5G. (Last Accessed 11 June 2018) - https://www.evolved-intelligence.com/events-and-media/blogs/there-s-no-time-like-the-present-to-secure-5g.

[57] Fajardo, V., Arkko, J., Loughney, J., and Zorn, G. (2012). Diameter Base Protocol. RFC 6733 (Proposed Standard). Updated by RFC 7075.

[58] Falowo, O. E. and Chan, H. A. (2005). AAA and Mobility Management in UMTSWLAN Interworking. In *Proc. 12th International Conference on Telecommications (ICT), Cape Town*. Citeseer.

[59] Farrell, S. (2018). Lpwan overview. Internet-Draft draft-ietf-lpwan-overview-10, Internet Engineering Task Force. Work in Progress.

[60] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. (1999). Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard). Obsoleted by RFCs 7230, 7231, 7232, 7233, 7234, 7235, updated by RFCs 2817, 5785, 6266, 6585.

[61] Forsberg, D., Ohba, Y., Patil, B., Tschofenig, H., and Yegin, A. (2008). Protocol for Carrying Authentication for Network Access (PANA). RFC 5191 (Proposed Standard). Updated by RFC 5872.

[62] Frankel, S., Eydt, B., Owens, L., and Scarfone, K. (2007). Establishing wireless robust security networks: a guide to IEEE 802.11 i. *National Institute of Standards and Technology*.

[63] Friel, O., Barnes, R., Pritikin, M., Tschofenig, H., and Baugher, M. (2018). Application-layer tls (atls). Internet-Draft draft-I-D.draft-friel-tls-atls, Internet Engineering Task Force. Work in Progress.

[64] Garcia, D., Garcia, S. N. M., and Lopez, R. (2016a). Application layer security for coap using the (d)tls record layer. Internet-Draft draft-I-D.draft-garcia-core-app-layer-sec-with-dtls-record, Internet Engineering Task Force. Work in Progress.

[65] Garcia, D., Lopez, R., Kandasamy, A., and Pelov, A. (2016b). LoRaWAN Authentication in RADIUS. Internet-Draft draft-I-D.draft-garcia-radext-radius-lorawan, Internet Engineering Task Force. Work in Progress.

[66] Garcia-Carrillo, D. and Marin-Lopez, R. (2016). Lightweight CoAP-Based Bootstrapping Service for the Internet of Things. *Sensors*, 16(3):358.

[67] Garcia-Carrillo, D., Marin-Lopez, R., Kandasamy, A., and Pelov, A. (2017). A CoAP-Based Network Access Authentication Service for Low-Power Wide Area Networks: LO-CoAP-EAP. *Sensors*, 17(11).

[68] Garcia-Morchon, O., Keoh, S. L., Kumar, S., Moreno-Sanchez, P., Vidal-Meca, F., and Ziegeldorf, J. H. (2013). Securing the ip-based internet of things with hip and dtls. In *Proceedings of the Sixth ACM Conference on Security and privacy in wireless and mobile networks*, pages 119–124. ACM.

[69] Garcia-Morchon, O., Kumar, S., and Sethi, M. (2017). State of the art and challenges for the internet of things. Internet-Draft draft-I-D.draft-irtf-t2trg-iot-seccons, Internet Engineering Task Force. Work in Progress.

[70] Garcia-Morchon, O., Kumar, S., and Sethi, M. (2018). State-of-the-art and challenges for the internet of things security. Internet-Draft draft-irtf-t2trg-iot-seccons-15, Internet Engineering Task Force. Work in Progress.

[71] Gomez, C., Oller, J., and Paradells, J. (2012a). Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology. *Sensors*, 12(9):11734–11753.

[72] Gomez, C., Oller, J., and Paradells, J. (2012b). Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology. *Sensors*, 12(9):11734.

[73] Granlund, D., Åhlund, C., and Holmlund, P. (2015). EAP-Swift: An efficient authentication and key generation mechanism for resource constrained WSNs. *International Journal of Distributed Sensor Networks*, 11(4):460914.

[74] Granlund, D. and Åhlund, C. (2011). A Scalability Study of AAA Support in Heterogeneous Networking Environments with Global Roaming Support. In *2011 IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 488–493.

[75] Gubbi, J., Buyya, R., Marusic, S., and Palaniswami, M. (2013). Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions. *Future Gener. Comput. Syst.*, 29(7):1645–1660.

[76] Hardjono, T. and Smith, N. (2016). Fluffy: Simplified key exchange for constrained environments. Internet-Draft draft-hardjono-ace-fluffy-03, Internet Engineering Task Force. Work in Progress.

[77] Hardt, D. (2012). The OAuth 2.0 Authorization Framework. RFC 6749 (Proposed Standard).

[78] Hawkins, S., Yen, D. C., and Chou, D. C. (2000). Awareness and challenges of Internet security. *Information Management & Computer Security*, 8(3):131–143.

[79] He, A. and sarikaya-core sbootstrapping, B. I.-D. (2015 - work in progressa). Iot security bootstrapping: Survey and design considerations. Internet-Draft draft-he-6lo-analysis-iot-sbootstrapping-00, IETF Secretariat. https://tools.ietf.org/html/draft-he-6lo-analysis-iot-sbootstrapping-00.

[80] He, A. and sarikaya-core sbootstrapping, B. I.-D. (2015 - work in progressb). Iot security bootstrapping: Survey and design considerations. Internet-Draft draft-he-6lo-analysis-iot-sbootstrapping-00, IETF Secretariat. https://tools.ietf.org/html/draft-he-6lo-analysis-iot-sbootstrapping-00.txt.

[81] Heer, T., Garcia-Morchon, O., Hummen, R., Keoh, S. L., Kumar, S. S., and Wehrle, K. (2011). Security challenges in the ip-based internet of things. *Wireless Personal Communications*, 61(3):527–542.

[82] Hennebert, C. and Santos, J. D. (2014). Security Protocols and Privacy Issues into 6LoWPAN Stack: A Synthesis. *IEEE Internet of Things Journal*, 1(5):384–398.

[83] Hernandez-Ramos, J. L., Carrillo, D. G., Marín-López, R., and Skarmeta, A. F. (2015a). Dynamic security credentials PANA-based provisioning for IoT smart objects. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pages 783–788.

[84] Hernandez-Ramos, J. L., Moreno, M. V., Bernabe, J. B., Carrillo, D. G., and Skarmeta, A. F. (2015b). SAFIR: Secure access framework for IoT-enabled services on smart buildings. *Journal of Computer and System Sciences*, 81(8):1452 – 1463.

[85] Hernández-Ramos, J. L., Pawlowski, M. P., Jara, A. J., Skarmeta, A. F., and Ladid, L. (2015). Toward a Lightweight Authentication and Authorization Framework for Smart Objects. *IEEE Journal on Selected Areas in Communications*, 33(4):690–702.

[86] Hogberg, J. (2011). Mobile provided identity authentication on the web. *W3C Workshop on Identity in the Browser*.

[87] Hossain, M. M., Fotouhi, M., and Hasan, R. (2015). Towards an analysis of security issues, challenges, and open problems in the internet of things. In *Services (SERVICES), 2015 IEEE World Congress on*, pages 21–28. IEEE.

[88] Housley, R. and Aboba, B. (2007). Guidance for Authentication, Authorization, and Accounting (AAA) Key Management. RFC 4962 (Best Current Practice).

[89] Howlett, J., Hartman, S., and Perez-Mendez, A. (2016). A RADIUS Attribute, Binding, Profiles, Name Identifier Format, and Confirmation Methods for the Security Assertion Markup Language (SAML). RFC 7833 (Proposed Standard).

[90] Hummen, R., Wirtz, H., Ziegeldorf, J. H., Hiller, J., and Wehrle, K. (2013). Tailoring end-to-end IP security protocols to the Internet of Things. In *2013 21st IEEE International Conference on Network Protocols (ICNP)*, pages 1–10.

[91] Ian Poole (2016). *Weightless Wireless M2M White Space Communications*. Available Online: (last access on August 2016) http://www.radio-electronics.com/info/wireless/weightless-m2m-white-space-wireless-communications/basics-overview.php.

[92] IEEE Computer Society (2006). *Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specification for Low-Rate Wireless Personal Area Networks (LR-WPANs)*. IEEE Std 802.15.4.

[93] IETF (2007). 3gpp ts 33.220 : Generic authentication architecture (gaa); generic bootstrapping architecture (gba). Technical report. http://www.3gpp.org/DynaReport/33220.htm.

[94] IETF (2012). CoRE Working Group. https://datatracker.ietf.org/wg/core/documents/.

[95] IETF (2013). 6tisch working group. https://datatracker.ietf.org/wg/6tisch/documents/.

[96] IETF (2014). ACE Working Group.

[97] IETF (2015). 6BAND (6lo Bootstrapping, Access for Networked Devices) Mailin List. https://www.ietf.org/mailman/listinfo/6band.

[98] Ishaq, I., Carels, D., Teklemariam, G. K., Hoebeke, J., Abeele, F. V. d., Poorter, E. D., Moerman, I., and Demeester, P. (2013). Ietf standardization in the field of the internet of things (iot): a survey. *Journal of Sensor and Actuator Networks*, 2(2):235–287.

[99] Jain, R. (1991). *The Art of Comp Systems Perform Analysis: Techniques for Experimental Design, Measurement, Simulation and Modelling*. Wiley.

[100] Jing, Q., Vasilakos, A. V., Wan, J., Lu, J., and Qiu, D. (2014). Security of the Internet of Things: perspectives and challenges. *Wireless Networks*, 20(8):2481–2501.

[101] Jose L. Hernandez-Ramos, Dan Garcia-Carrillo, A. S. F. G. L. C. J.-M. B. M. B. (2016). Smartie: a secure platform for smart cities and iot. In Benjamin Aziz, Alvaro Arenas, B. C., editor, *Engineering Secure Internet of Things Systems*, chapter 4, pages 266–290. Institution of Engineering and Technology.

[102] Kandasamy, A., Lopez, R., Garcia, D., and Pelov, A. (2016). LoRaWAN Authentication in Diameter. Internet-Draft draft-I-D.draft-garcia-dime-diameter-lorawan, Internet Engineering Task Force. Work in Progress.

[103] Kang, N., Oh, S.-H., and Yoon, S. (2014). Secure initial-key reconfiguration for resource constrained devices. Internet-Draft draft-I-D.draft-kang-core-secure-reconfiguration, Internet Engineering Task Force. Work in Progress.

[104] Kato, T. (2015). Standardization and Certification Process for "Wi-SUN" Wireless Communication Technology. Technical Review 23, Anritsu.

[105] Kaufman, C., Hoffman, P., Nir, Y., and Eronen, P. (2010). Internet Key Exchange Protocol Version 2 (IKEv2). RFC 5996 (Proposed Standard). Obsoleted by RFC 7296, updated by RFCs 5998, 6989.

[106] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and Kivinen, T. (2014). Internet Key Exchange Protocol Version 2 (IKEv2). RFC 7296 (Internet Standard). Updated by RFCs 7427, 7670.

[107] Kent, S. and Seo, K. (2005). Security Architecture for the Internet Protocol. RFC 4301 (Proposed Standard). Updated by RFCs 6040, 7619.

[108] Keoh, S. L., Garcia-Morchon, O., and Kumar, S. S. (2014). Dtls relay for constrained environments. Internet-Draft draft-I-D.draft-kumar-dice-dtls-relay, Internet Engineering Task Force. Work in Progress.

[109] Khan, R., Khan, S. U., Zaheer, R., and Khan, S. (2012). Future internet: the internet of things architecture, possible applications and key challenges. In *Frontiers of Information Technology (FIT), 2012 10th International Conference on*, pages 257–260. IEEE.

[110] Khorov, E., Lyakhov, A., Krotov, A., and Guschin, A. (2015). A survey on {IEEE} 802.11ah: An enabling networking technology for smart cities. *Computer Communications*, 58:53 – 69. Special Issue on Networking and Communications for Smart Cities.

[111] Kivinen, T. (2016). Minimal Internet Key Exchange Version 2 (IKEv2) Initiator Implementation. RFC 7815 (Informational).

[112] Korhonen, J. (2012). Applying Generic Bootstrapping Architecture for use with Constrained Devices - IAB Workshop on 'Smart Object Security'.

[113] Kortuem, G., Kawsar, F., Sundramoorthy, V., and Fitton, D. (2010). Smart objects as building blocks for the internet of things. *IEEE Internet Computing*, 14(1):44–51.

[114] Kovatsch, M., Bergmann, O., and Bormann, C. (2017a). Coap implementation guidance. Internet-Draft draft-ietf-lwig-coap-05, Internet Engineering Task Force. Work in Progress.

[115] Kovatsch, M., Bergmann, O., and Bormann, C. (2017b). Coap implementation guidance. Internet-Draft draft-I-D.draft-ietf-lwig-coap, Internet Engineering Task Force. Work in Progress.

[116] Leavitt, N. (2010). Researchers fight to keep implanted medical devices safe from hackers. *Computer*, 43(8):11–14.

[117] Leo, M., Battisti, F., Carli, M., and Neri, A. (2014). A federated architecture approach for Internet of Things security. In *Euro Med Telco Conference (EMTC), 2014*, pages 1–5. IEEE.

[118] Levi, D., Meyer, P., and Stewart, B. (2002). Simple Network Management Protocol (SNMP) Applications. RFC 3413 (Internet Standard).

[119] Liu, W., Zhong, J., and Lin, J. (2013). Secure Access Scheme Research and Design Based on the Internet of Things. In *Applied Mechanics and Materials*, volume 278, pages 1818–1821. Trans Tech Publ.

[120] livinginternet.com (2000). The Internet Toaster. Technical report. [Online; accessed 02-May-2018].

[121] López, G., Cánovas, O., Gómez, A. F., Jiménez, J. D., and Marín, R. (2007). A network access control approach based on the aaa architecture and authorization attributes. *Journal of Network and Computer Applications*, 30(3):900–919.

[122] Lopez, R. and Garcia, D. (2017). Eap-based authentication service for coap. Internet-Draft draft-I-D.draft-marin-ace-wg-coap-eap, Internet Engineering Task Force. Work in Progress.

[123] Mahmoud, R., Yousuf, T., Aloul, F., and Zualkernan, I. (2015). Internet of things (IoT) security: Current status, challenges and prospective measures. In *Internet Technology and Secured Transactions (ICITST), 2015 10th International Conference for*, pages 336–341. IEEE.

[124] Manyika, J. (2015). *The Internet of Things: Mapping the value beyond the hype*. McKinsey Global Institute.

[125] McGrew, D. (2008). An Interface and Algorithms for Authenticated Encryption. RFC 5116 (Proposed Standard).

[126] Meca, F. V., Ziegeldorf, J. H., Sanchez, P. M., Morchon, O. G., Kumar, S. S., and Keoh, S. L. (2013). Hip security architecture for the ip-based internet of things. *2013 27th International Conference on Advanced Information Networking and Applications Workshops*, 0:1331–1336. http://doi.ieeecomputersociety.org/10.1109/WAINA.2013.158.

[127] Mills, A. (2013). CoAP Implementation that Focuses on Simplicity. Software, Github. (Last Accessed 23 November 2016) - https://github.com/staropram/cantcoap.

[128] Minaburo, A. and Toutain, L. (2018). Lpwan static context header compression (schc) for coap. Internet-Draft draft-I-D.draft-ietf-lpwan-coap-static-context-hc, Internet Engineering Task Force. Work in Progress.

[129] Minaburo, A., Toutain, L., Gomez, C., Paradells, J., and Crowcroft, J. (2016). Lpwan survey and gap analysis. Internet-Draft draft-I-D.draft-minaburo-lpwan-gap-analysis, Internet Engineering Task Force. Work in Progress.

[130] Moreno-Sanchez, P., Marin-Lopez, R., and Vidal-Meca, F. (2014a). An open source implementation of the protocol for carrying authentication for network access: OpenPANA. *IEEE Network*, 28(2):49–55.

[131] Moreno-Sanchez, P., Marin-Lopez, R., and Vidal-Meca, F. (2014b). An open source implementation of the protocol for carrying authentication for network access: Openpana. *Network, IEEE*, 28(2):49–55.

[132] Moskowitz, R. and Hummen, R. (2017). Hip diet exchange (dex). Internet-Draft draft-ietf-hip-dex-06, Internet Engineering Task Force. Work in Progress.

[133] Moskowitz, R. and Nikander, P. (2006). Host Identity Protocol (HIP) Architecture. RFC 4423 (Informational).

[134] Moyer, B. (2015). Low power, wide area. a survey of longer-range iot wireless protocols. article, eejournal.com. http://www.eejournal.com/archives/articles/20150907-lpwa/.

[135] Nakhjiri, M. and Nakhjiri, M. (2005). *AAA and network security for mobile access: RADIUS, Diameter, EAP, PKI and IP mobility*. John Wiley & Sons.

[136] Naranjo, J., Orduna, P., Gómez-Goiri, A., López-de Ipina, D., and Casado, L. (2012). Lightweight user access control in energy-constrained wireless network services. In *Ubiquitous Computing and Ambient Intelligence*, pages 33–41. Springer.

[137] NEC (2018). Making 5G a Reality. (Last Accessed 11 June 2018) - https://www.nec.com/en/global/solutions/nsp/5g_vision/doc/wp2018ar.pdf.

[138] Nelson, D. (2011). Crypto-Agility Requirements for Remote Authentication Dial-In User Service (RADIUS). RFC 6421 (Informational).

[139] Nieminen, J., Savolainen, T., Isomaki, M., Patil, B., Shelby, Z., and Gomez, C. (2015). IPv6 over BLUETOOTH(R) Low Energy. RFC 7668 (Proposed Standard).

[140] Nikander, P., Ylitalo, J., and Wall, J. (2003). Integrating security, mobility and multi-homing in a hip way. In *NDSS*, volume 3, pages 6–7.

[141] Nir, Y., Tschofenig, H., Deng, H., and Singh, R. (2010). A Childless Initiation of the Internet Key Exchange Version 2 (IKEv2) Security Association (SA). RFC 6023 (Experimental).

[142] Noll, J. and Chowdhury, M. M. (2011). 5G: Service continuity in heterogeneous environments. *Wireless Personal Communications*, 57(3):413–429.

[143] Nordrum, A. (2016). Popular Internet of Things Forecast of 50 Billion Devices by 2020 Is Outdated. Technical report. [Online; accessed 02-May-2018].

[144] O'Flynn, C. P., Sarikaya, B., Ohba, Y., Cao, Z., and Cragie, R. (2010 - work in progress). Security bootstrapping of resource-constrained devices. Internet-Draft draft-oflynn-core-bootstrapping-03, IETF Secretariat. https://tools.ietf.org/html/draft-oflynn-core-bootstrapping-03.

[145] Osterlind, F., Dunkels, A., Eriksson, J., Finne, N., and Voigt, T. (2006a). Cross-level sensor network simulation with cooja. *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*, pages 641–648.

[146] Osterlind, F., Dunkels, A., Eriksson, J., Finne, N., and Voigt, T. (2006b). Cross-Level Sensor Network Simulation with COOJA. In *Proceedings. 2006 31st IEEE Conference on Local Computer Networks*, pages 641–648.

[147] Palattella, M. R., Accettura, N., Vilajosana, X., Watteyne, T., Grieco, L. A., Boggia, G., and Dohler, M. (2013). Standardized protocol stack for the internet of (important) things. *Communications Surveys & Tutorials, IEEE*, 15(3):1389–1406.

[148] Patel, A. and Giaretta, G. (2006). Problem Statement for bootstrapping Mobile IPv6 (MIPv6). RFC 4640 (Informational).

[149] Pathak, P. (2016). The Internet of Things: Study of Security and Privacy Considerations. *challenge*, 2(6):12.

[150] Pawlowski, M. P., Jara, A. J., and Ogorzalek, M. J. (2015). EAP for IoT: More Efficient Transport of Authentication Data–TEPANOM Case Study. In *Advanced Information Networking and Applications Workshops (WAINA), 2015 IEEE 29th International Conference on*, pages 694–699. IEEE.

[151] Petrić, T., Goessens, M., Nuaymi, L., Toutain, L., and Pelov, A. (2016). Measurements, performance and analysis of lora fabian, a real-world implementation of lpwan. In *2016 IEEE 27th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pages 1–7.

[152] Pritikin, M., Richardson, M., Behringer, M. H., Bjarnason, S., and Watsen, K. (2018). Bootstrapping remote secure key infrastructures (brski). Internet-Draft draft-I-D.draft-ietf-anima-bootstrapping-keyinfra, Internet Engineering Task Force. Work in Progress.

[153] Pritikin, M., Yee, P., and Harkins, D. (2013). Enrollment over Secure Transport. RFC 7030 (Proposed Standard).

[154] Pruss, R. and Zorn, G. (2010 - work in progress). Eap authentication extensions for the dynamic host configuration protocol for broadband. Internet-Draft draft-pruss-dhcp-auth-dsl-07, IETF Secretariat. https://tools.ietf.org/html/draft-pruss-dhcp-auth-dsl-07.

[155] Quentin Stafford-Fraser (1995). The Trojan Room Coffee Pot. Technical report. [Online; accessed 02-May-2018].

[156] R. Dantu, G. Clothier, and A. Atri (2007). *EAP Methods for Wireless Networks. Computer Standards Interfaces*, 29(3)(3):289–301.

[157] R. Marin-Lopez, F. Pereniguez-Garcia, F. Gomez-Skarmeta, and Y. Ohba (2012). *Network Access Security for the Internet: Protocol for Carrying Authentication for Network Access. IEEE Communications Magazine.*

[158] Rao, S., Chendanda, D., Deshpande, C., and Lakkundi, V. (2015). Implementing lwm2m in constrained iot devices. In *Wireless Sensors (ICWiSe), 2015 IEEE Conference on*, pages 52–57.

[159] Ratilainen, A. (2016). Nb-iot characteristics. Internet-Draft draft-I-D.draft-ratilainen-lpwan-nb-iot, Internet Engineering Task Force. Work in Progress.

[160] Raza, S., Voigt, T., and Jutvik, V. (2012). Lightweight IKEv2: a key management solution for both the compressed IPsec and the IEEE 802.15. 4 security. In *Proceedings of the IETF workshop on smart object security*, volume 23. Citeseer.

[161] Recordon, D. and Reed, D. (2006). OpenID 2.0: a platform for user-centric identity management. In *Proceedings of the second ACM workshop on Digital identity management*, pages 11–16. ACM.

[162] Rescorla, E. and Modadugu, N. (2012). Datagram Transport Layer Security Version 1.2. RFC 6347 (Proposed Standard). Updated by RFCs 7507, 7905.

[163] Richardson, L. and Ruby, S. (2008). *RESTful web services*. " O'Reilly Media, Inc.".

[164] Richardson, M. and Damm, B. (2018). 6tisch Zero-Touch Secure Join protocol. Internet-Draft draft-ietf-6tisch-dtsecurity-zerotouch-join-02, Internet Engineering Task Force. Work in Progress.

[165] Richardson, R. (2015). Iot discovery and federation controls lacking. http://searchsecurity.techtarget.com/news/4500244846/IoT-discovery-and-federation-controls-lacking.

[166] Rigney, C. (2000). RADIUS Accounting. RFC 2866 (Informational). Updated by RFCs 2867, 5080, 5997.

[167] Rigney, C., Willens, S., Rubens, A., and Simpson, W. (2000). Remote Authentication Dial In User Service (RADIUS). RFC 2865 (Draft Standard). Updated by RFCs 2868, 3575, 5080, 6929, 8044.

[168] Roman, R., Zhou, J., and Lopez, J. (2013). On the features and challenges of security and privacy in distributed internet of things. *Computer Networks*, 57:2266–2279.

[169] Rosenberg, J. B. and Remy, D. L. (2004). *Securing web services with WS-security: Demystifying WS-security, WS-policy, SAML, XML signature, and XML encryption*. Sams.

[170] Sadeghi, A.-R., Wachsmann, C., and Waidner, M. (2015). Security and privacy challenges in industrial internet of things. In *Proceedings of the 52nd annual design automation conference*, page 54. ACM.

[171] Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and Mortimore, C. (2014). OpenID Connect Core 1.0 incorporating errata set 1. *The OpenID Foundation, specification*.

[172] Salkintzis, A. K. (2004). Interworking techniques and architectures for wlan/3g integration toward 4g mobile data networks. *Wireless Communications, IEEE*, 11(3):50–61.

[173] Salowey, J., Dondeti, L., Narayanan, V., and Nakhjiri, M. (2008). Specification for the Derivation of Root Keys from an Extended Master Session Key (EMSK). RFC 5295 (Proposed Standard).

[174] Sanchez, P. M., Lopez, R. M., and Skarmeta, A. F. G. (2013a). Panatiki: A network access control implementation based on pana for iot devices. *Sensors*, 13(11):14888.

[175] Sanchez, P. M., Lopez, R. M., and Skarmeta, A. F. G. (2013b). PANATIKI: A Network Access Control Implementation Based on PANA for IoT Devices. *Sensors*, 13(11):14888.

[176] Sanchez-Iborra, R. and Cano, M.-D. (2016). State of the Art in LP-WAN Solutions for Industrial IoT Services. *Sensors*, 16(5):708.

[177] Sanchez-Iborra, R., Sanchez-Gomez, J., and Skarmeta, A. (2018). Evolving iot networks by the confluence of mec and lp-wan paradigms. *Future Generation Computer Systems*.

[178] Sarikaya, B., Cragie, R., Moskowitz, R., Ohba, Y., and Cao, Z. (2013). Security bootstrapping solution for resource-constrained devices. Internet-Draft draft-I-D.draft-sarikaya-core-sbootstrapping, Internet Engineering Task Force. Work in Progress.

[179] Sarikaya, B., Sethi, M., and Garcia, D. (2018). Secure IoT Bootstrapping: A Survey. Internet-Draft draft-sarikaya-t2trg-sbootstrapping-04, Internet Engineering Task Force. Work in Progress.

[180] sarikaya-core sbootstrapping, B. I.-D. (2013). Secure bootstrapping solution for resource-constrained devices. Internet-Draft draft-I-D.draft-sarikaya-6lo-bootstrapping-solution, Internet Engineering Task Force. Work in Progress.

[181] Schaad, J. (2017). CBOR Object Signing and Encryption (COSE). RFC 8152.

[182] Schmertmann, L., Hartke, K., and Bormann, C. (2014). Codtls: Dtls handshakes over coap. Internet-Draft draft-I-D.draft-schmertmann-dice-codtls, Internet Engineering Task Force. Work in Progress.

[183] Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and Tschofenig, H. (2018). Authentication and authorization for constrained environments (ace) using the oauth 2.0 framework (ace-oauth). Internet-Draft draft-ietf-ace-oauth-authz-12, Internet Engineering Task Force. Work in Progress.

[184] Selander, G., Mattsson, J., and Palombini, F. (2018a). Ephemeral diffie-hellman over cose (edhoc). Internet-Draft draft-I-D.draft-selander-ace-cose-ecdhe, Internet Engineering Task Force. Work in Progress.

[185] Selander, G., Mattsson, J., Palombini, F., and Seitz, L. (2018b). Object security for constrained restful environments (oscore). Internet-Draft draft-I-D.draft-ietf-core-object-security, Internet Engineering Task Force. Work in Progress.

[186] Selander, G., Mattsson, J., Palombini, F., and Seitz, L. (2018c). Object security for constrained restful environments (oscore). Internet-Draft draft-I-D.draft-ietf-core-object-security, Internet Engineering Task Force. Work in Progress.

[187] Semtech (2016). Semtech - SX1272 LoRa Calculator Tool. Software, Semtech. Available Online (last access on March 2016) http://www.semtech.com.

[188] Shelby, Z. and Bormann, C. (2011). *6LoWPAN: The wireless embedded Internet*, volume 43. John Wiley & Sons.

[189] Shelby, Z. and Chauvenet, C. (2012). The ipso application framework draft-ipso-app-framework-04. *IPSO Alliance, Interop Committee*.

[190] Shelby, Z., Hartke, K., and Bormann, C. (2014). The Constrained Application Protocol (CoAP). RFC 7252 (Proposed Standard). Updated by RFC 7959.

[191] Shelby, Z., Koster, M., Bormann, C., der Stok, P. V., and Amsüss, C. (2018). Core resource directory. Internet-Draft draft-ietf-core-resource-directory-13, Internet Engineering Task Force. Work in Progress.

[192] Sigfox, S. (2016). Sigfox one network a billion dreams. m2m and iot redefined through cost effective and energy optimized connectivity. (white paper). Available online: (Last Accessed: 22 August 2016).

[193] Simon, D., Aboba, B., and Hurst, R. (2008). The EAP-TLS Authentication Protocol. RFC 5216 (Proposed Standard).

[194] Soheil, Mahdizadeh, S., Dustdar, S., Behinaein, N., and Rahimzadeh, R. (2015 - work in progress). Diameter of things (dot): A protocol for real-time telemetry of iot applications. Internet-Draft draft-tuwien-dsg-diameterofthings-01, IETF Secretariat. https://tools.ietf.org/html/draft-tuwien-dsg-diameterofthings-01.

[195] Song, J., Poovendran, R., Lee, J., and Iwata, T. (2006a). The Advanced Encryption Standard-Cipher-based Message Authentication Code-Pseudo-Random Function-128 (AES-CMAC-PRF-128) Algorithm for the Internet Key Exchange Protocol (IKE). RFC 4615 (Proposed Standard).

[196] Song, J., Poovendran, R., Lee, J., and Iwata, T. (2006b). The AES-CMAC Algorithm. RFC 4493 (Informational).

[197] Sornin, N., Luis, M., Eirich, T., Kramp, T., and Hersent, O. (2015). LoRaWAN™ Specifications. *LoRa™ Alliance*.

[198] Stanley, D., Walker, J., and Aboba, B. (2005). Extensible Authentication Protocol (EAP) Method Requirements for Wireless LANs. RFC 4017 (Informational).

[199] Stewart, R. (2007). Stream Control Transmission Protocol. RFC 4960 (Proposed Standard). Updated by RFCs 6096, 6335, 7053.

[200] Technologies, P. (2017). NEXT GENERATION NETWORKS, NEXT LEVEL CYBERSECURITY PROBLEMS. (Last Accessed 11 June 2018) - https://www.ptsecurity.com/upload/corporate/ww-en/analytics/diameter_research.pdf.

[201] Tian, L. (2012). Lightweight M2M (OMA LWM2M). *OMA device management working group (OMA DM WG), Open Mobile Alliance (OMA)*.

[202] Tschofenig, H., Arkko, J., Thaler, D., and McPherson, D. (2015). Architectural Considerations in Smart Object Networking. RFC 7452 (Informational).

[203] University, C. M. (2000). The "Only" Coke Machine on the Internet. Technical report. [Online; accessed 02-May-2018].

[204] Vacca, J. R. (2012). *Computer and information security handbook*. Newnes.

[205] Vollbrecht, J., Eronen, P., Petroni, N., and Ohba, Y. (2005). State Machines for Extensible Authentication Protocol (EAP) Peer and Authenticator. RFC 4137 (Informational).

[206] Vučinić, M., Simon, J., Pister, K., and Richardson, M. (2018). Minimal security framework for 6tisch. Internet-Draft draft-I-D.draft-ietf-6tisch-minimal-security, Internet Engineering Task Force. Work in Progress.

[207] Vucinic, M., Tourancheau, B., Watteyne, T., Rousseau, F., Duda, A., Guizzetti, R., and Damon, L. (2015). Dtls performance in duty-cycled networks. *arXiv preprint arXiv:1507.05810*.

[208] Wang, Q., Vilajosana, X., Watteyne, T., Sudhaakar, R., and Zand, P. (2015 - work in progress). Transporting coap messages over ieee802.15.4e information elements. Internet-Draft draft-wang-6tisch-6top-coapie-01, IETF Secretariat. https://tools.ietf.org/html/draft-wang-6tisch-6top-coapie-01.txt.

[209] Watteyne, T., Palattella, M., and Grieco, L. (2015). Using IEEE 802.15.4e Time-Slotted Channel Hopping (TSCH) in the Internet of Things (IoT): Problem Statement. RFC 7554 (Informational).

[210] Weyn, M., Ergeerts, G., Berkvens, R., Wojciechowski, B., and Tabakov, Y. (2015). DASH7 alliance protocol 1.0: Low-power, mid-range sensor and actuator communication. In *Standards for Communications and Networking (CSCN), 2015 IEEE Conference on*, pages 54–59.

[211] Whiting, D., Housley, R., and Ferguson, N. (2003). Counter with CBC-MAC (CCM). RFC 3610 (Informational).

[212] Wierenga, K. and Florio, L. (2005). Eduroam: past, present and future. *Computational methods in science and technology*, 11(2):169–173.

[213] Xiong, X., Zheng, K., Xu, R., Xiang, W., and Chatzimisios, P. (2015). Low power wide area machine-to-machine networks: key techniques and prototype. *Communications Magazine, IEEE*, 53(9):64–71.

[214] Yegin, A. and Shelby, Z. (2011 - work in progress). Coap security options. Internet-Draft draft-I-D.draft-yegin-coap-security-options, IETF Secretariat. https://tools.ietf.org/html/draft-I-D.draft-yegin-coap-security-options.

[215] Zhang, Z.-K., Cho, M. C. Y., Wang, C.-W., Hsu, C.-W., Chen, C.-K., and Shieh, S. (2014). IoT security: ongoing challenges and research opportunities. In *Service-Oriented Computing and Applications (SOCA), 2014 IEEE 7th International Conference on*, pages 230–234. IEEE.

[216] Zhao, K. and Ge, L. (2013). A survey on the internet of things security. In *Computational Intelligence and Security (CIS), 2013 9th International Conference on*, pages 663–667. IEEE.

[217] ZigBee Alliance (2014). ZigBee IP specification, ZigBee document 095023r34.

[218] Zorn, G., Zhang, T., Walker, J., and Salowey, J. (2011). Cisco Vendor-Specific RADIUS Attributes for the Delivery of Keying Material. RFC 6218 (Informational).