Raghavendra Mudugodu Seetarama

Secure Device Bootstrapping with the Nimble Out of Band Authentication Protocol

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 14.05.2017

Thesis supervisor:

Prof. Tuomas Aura

Thesis advisor:

Mohit Sethi, D.Sc. (Tech.)

Aalto University
School of Electrical Engineering

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author: Raghavendra Mudugodu Seetarama

Title: Secure Device Bootstrapping with the Nimble Out of Band Authentication Protocol

Date: 14.05.2017 Language: English Number of pages: 8+63

Department of Computer Science

Professorship: Secure Systems Code: S-55

Supervisor: Prof. Tuomas Aura

Advisor: Mohit Sethi, D.Sc. (Tech.)

The smart personal and business appliances which form the Internet of Things are expected to become ubiquitous and to make our daily life more convenient. Most of these devices are connected though wireless networks to cloud-based online services. However, such devices may be vulnerable to various attacks which could compromise the users' security and privacy and even cause physical harm. Therefore, securing the network connection for the devices is of utmost importance.
In order to secure the network connections, the devices need to be configured with the necessary keys and other connection parameters. There is not yet any widely adopted generic solution for this secure bootstrapping. One proposed solution is out-of-band (OOB) authentication with a protocol called EAP-NOOB, which is a new method for the EAP and IEEE 802.1X authentication framework.
The goal of this thesis is to build a prototype of the EAP-NOOB protocol and deploy the prototype to test it with the real-world scenarios. The protocol requires no a-priori information either about the device or the user is necessary for the bootstrapping. Instead, the user's ownership of the device is established during the bootstrapping process. The protocol was implemented both by adding support for the new EAP method into existing open-source software, the commonly used WPA_Supplicant and Hostapd packages. We also implemented a web interface for the back-end authentication server, which works in tandem with the AAA server, and out-of-band channels based on dynamic QR codes and NFC tags.
We used the prototype to test and demonstrate the EAP-NOOB protocol, including its usability and authentication latency. The bootstrapping procedure can be completed in less than a minute in most cases. The main results of the project are the EAP-NOOB implementation and various improvements and clarifications to the protocol specification. These results are an essential part of the protocol standardization process at IETF.

Keywords: IoT, secure bootstrapping, EAP, out-of-band authentication, EAP-NOOB

Preface

First and foremost, I would like to thank professor Tuomas Aura for providing me the opportunity to work in his research group. I am grateful for all the guidance that he provided and my sincere gratitude for taking the team to IETF-96, Berlin.

I would like to thank Mohit Sethi for his continuous support and for all his timely advice. Most importantly, i am grateful for all the technical guidance that i received.

I sincerely thank my teammate Shiva Prasad Thagadur Prakash for his contribution and hard work towards the project. Working with him had been a great learning and enjoyable experience.

Finally, this thesis would not have been possible without the encouragement and support from my family. I will take this opportunity to express my gratitude to them.

Otaniemi, 14.05.2017

Raghavendra Mudugodu Seetarama

Contents

Abbreviations

AAA	Authentication Authorization and Accounting
AES	Advance Encryption System
AP	Access Point
API	Application Programme Interface
ASCII	American Standard Code for Information Interchange
BSD	Berkeley Software Distribution
CA	Certificate Authority
CCMP	Counter-Mode-CBC-MAC
DTLS	Datagram Transport Layer Security
EAP	Extensible Authentication Protocol
EAPOL	EAP over LAN
ECC	Elliptic Curve Cryptosystem
ESS	Extended Service Set
ESSID	Extended Service Set Identifier.
GSM	Global System for Mobile Communication
GUI	Graphical User Interface
IANA	Internet Assigned Numbers Authority
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IOT	Internet of Things
IP	Internet Protocol
ISP	Internet Service Provider
KDF	Key Derivation Function
LAN	Local Area Network
LTE	Long Term Evolution
MAC	Message Authentication Code
MIC	Message Integrity Check
MIT	Massachusetts Institute of Technology
MITM	Man In The Middle
NAI	Network Access Identifier
NDEF	NFC Data Exchange Format
NFC	Near Field Communication
NIST	National Institute of Standards and Technology
NOOB	Nimble Out-Of-Band
OOB	Out-Of-Band
PMK	Pairwise Master Key
PSK	Pre-Shared Key
PTK	Pairwise Transient Key
QR	Quick Response
RADIUS	Remote Authentication Dial-In User Service
RC4	Rivest Cipher 4
RFC	Request For Comment
SSID	Service Set Identifier

STA	Station
TKIP	Temporal Key Integrity Protocol
TLS	Transport Layer Security
URL	Universal Resource Locator.
WECA	Wireless Ethernet Compatibility Alliance
WEP	Wired Equivalent Privacy
Wi-Fi	Wireless Fidelity
WiMAX	Worldwide Interoperability for Microwave Access
WLAN	Wireless Local Area Network
WPA	Wi-Fi Protected Access
WPA2	Wi-Fi Protected Access 2
WPS	Wi-Fi Protected Setup

List of Figures

List of Tables

1 Introduction

1.1 Overview

The Internet of things, also known as IoT, is a new paradigm of technology which is making complex and efficient network of smart devices a reality. The ability to sense and control an environment with ease from a remote location makes everyday life more convenient. Furthermore, the convergence of Internet technology, data science, embedded systems and wireless technology has resulted in creating a solution which has the potential to change our way of living in the near future [1].

In the era of IoT, each connecting device is getting embed with some intelligence to interact with the controlling entity. Some credit should also be given to cheaper memory and microprocessors. It is anticipated that, in the near future IoT will put more devices onto the Internet. Even the traditional household devices like refrigerator and dishwasher will now be part of the Internet. The homes that we live to clothes that we wear [47], we will be surrounded by a cluster of connected smart devices. Apart from household environment, IoT also has a diverse set of application. From remote health monitoring [24] to industrial automation, application of IoT can be seen in any area of science. Hence, the network of smart devices will be omnipresent.

Although the IoT is expected to make our daily life more convenient, security of IoT devices when connected to Internet should be a topic of concern. Consider a smart home where appliances like microwave oven and refrigerators can be remotely controlled. Also, there are provisions to control indoor temperature and door locks for the smart home. Without the adequate security measures for the devices, an adversary may gain control over the devices. Now, actions such as unlocking a door or disabling power supply are possible for an adversary. In a worst case scenario, people living in the home may get physically harmed. It can certainly be stated that, irrespective of the deployed environment, adequate measures to secure the Internet connected devices is an absolute necessity

One of the convenient ways to connect devices to the Internet is through wireless access network, especially when the numbers of devices to be connected are more. Both at home and corporate environment, many appliances and devices gain access to the Internet through Wi-Fi access points. A connection to a Wi-Fi access point can be of open or secure type. To have a secure type of connection with an access point, either WPA2-Personal or WPA2-Enterprise security protocols will be used. However, to use either of the protocols, a user will first have to configure the access point to host a Wi-Fi network. Additionally, the user is also responsible for configuring the devices to have a secure Internet configuration. For an average user, it will be a challenge to configure each of the devices, considering that the devices will differ in their user interfaces. Also, after the configuration, monitoring and managing the device with adequate security will be equally challenging.

As part of configuration, each device has to be authenticated and it is done with the help of credentials. Usually, the credentials used for authentication are method dependent. In WPA2-Personal, authentication is done based on a pre-shared key, and the key is same for all the authenticating devices. However, WPA2-Enterprise inherently supports multiple authentication methods and hence credentials vary based on the used method of authentication. Unlike WPA-Personal, here, each authenticating device will have its own set of credentials and also offers a centralised governance for the devices. Hence, WPA2-Enterprise is considered to be more extensible.

Irrespective of the chosen protocol for authentication, a user will either have to pre-install or provide credentials at the time of authentication. Pre-installing credentials requires investment in both time and money, and it is definitely not scalable. Also, in the case of change of ownership of a device, the new user may not want to use same set of credentials, as there are no means to find if the existing credentials are compromised. Although the credentials are important for authentication, managing them for a large set of devices may become tedious.

As an alternative to user credential based authentication, one can use newly proposed authentication method named EAP-NOOB. The proposed method is a WPA2-Enterprise based authentication protocol and it is not dependent on any pre-installed credential or any pre-assigned identity for authentication. Additionally, the method can also be used to securely configure, monitor and manage all the devices of a user from a single platform. Now, considering the expected growth in the number of smart devices in the near future [4], a user friendly protocol to securely configure and manage the devices is expected to be very useful. In this thesis we will implement the proposed authentication and device management protocol EAP-NOOB. Also, we will analyze the protocol to verify its adequacy for user's needs in the future.

This thesis aims to make the existence of IoT devices on the Internet more secure. Also, it aims to make the activity of configuring IoT devices and managing them from a cloud environment user friendly. Activities planned in this thesis to achieve the set objectives can be listed as:

– Implement a prototype of EAP-NOOB protocol as per the latest specification [42].

– Confirm all the assumptions made by the specification by deploying the prototype for secure bootstrapping.

– Authenticating and configuring devices using the EAP-NOOB implementation to gain Internet and cloud connectivity.

– To build a single platform for easy management of all the authenticated IoT devices.

– Analyse and optimise the authentication latency on the implemented prototype.

Specifying the EAP-NOOB protocol is an ongoing research project. The specifications are based on both theoretical research and practical implementations. This thesis is done as part of the research group and it has contributed in specifying the protocol by developing a prototype, which provide means to test and analyse the protocol. To achieve the set objectives for this thesis, we have used several research methods on the protocol. The set of methods can be listed as:

– Experimental implementation of the EAP-NOOB protocol.

– Testing each of the protocol functionality.

– Detail analysis on the protocol security offerings.

– Redesigning the protocol message sequences to handle real world authentication scenarios.

The EAP-NOOB protocol is defined to be a generic protocol and applicable to IoT devices and appliances of any make. To make the protocol visible to large user community, and also to make devices interoperable with the protocol, EAP-NOOB is proposed to be standardized. A draft version of the protocol specification with the name *"Nimble out-of-band authentication for EAP (EAP-NOOB)"* [42] has been submitted to the Internet Engineering Task Force (IETF). This thesis is also done as part of the same research group which has specified the EAP-NOOB protocol.

1.2 Thesis Organisation

The rest of the thesis is structured as, in chapter 2 we will discuss all the background topics necessary for understanding the protocol. In chapter 3 we provide a detail explanation of EAP-NOOB protocol. Next, chapter 4 describes the complete protocol implementation. Later we will analyze the authentication latency in chapter 5. Finally, the concluding remarks will be mentioned in chapter 6.

2 Background

In this chapter, we introduce the protocols and concepts necessary to understand the EAP-NOOB protocol. At first, we provide a brief introduction about IoT security and secure bootstrapping. Next, we briefly discuss the available security protocols in Wi-Fi technology. Later in the chapter, we describe a few technologies related to secure configuration of a device. Finally, we will end this chapter with a short explanation of out-of-band channels.

2.1 IoT Security

Smart devices which can operate on the existing infrastructure are now available for affordable prices. For example, Amazon Dash[1], a Wi-Fi connected device available for less than $10, enables the user to restock their household-supply inventory just by a button press. With the prospect of our daily life filled with such Internet connected smart devices, it is absolutely necessary to focus on the security of the devices [28]. Security for Internet connected devices is always necessary, and now it is even more important because a security breach for an Internet-connected device can affect users physically.

The heterogeneous nature of IoT devices makes it difficult to have a single security solution across all the devices. Hence, ensuring security in IoT is often challenging. A constrained IoT device will only have limited memory and restricted computational power to execute large and complex security protocols. This will not be the case for high-end appliances which are as good as a desktop computer. However, irrespective of the capabilities of the device, authentication and authorization of the device should be done to identify the trusted devices and the back-end servers. Additionally, a secure connection between the device and the back-end cloud service should be established to protect the privacy of user data. Therefore, ensuring security of IoT devices over Internet is an absolute necessity. Out of many aspects which involves ensuring security of the IoT devices, this thesis only addresses ensuring security through bootstrapping.

An industry-wide common definition and understanding is still to be established for IoT. The network stack in a typical IoT device will have protocols from multiple standardization bodies. IEEE (Institute of Electrical and Electronics Engineers), 3GPP (Third generation partnership project), Wi-Fi (Wireless Fidelity) alliance and Bluetooth special interest group are the prominent standardization bodies or consortiums, individually developing standards for link layer communication. Each of these working bodies will include security solution in their link layer standard for an IoT environment. For example, the Wi-Fi alliance is a group of companies certifying Wi-Fi enabled devices and it is specifying a new Device Provisioning Protocol (DPP) to securely configure diverse set of devices into a Wi-Fi network [31]. IEEE

[1]`https://www.amazon.com/Dash-Buttons/b?ie=UTF8&node=10667898011`

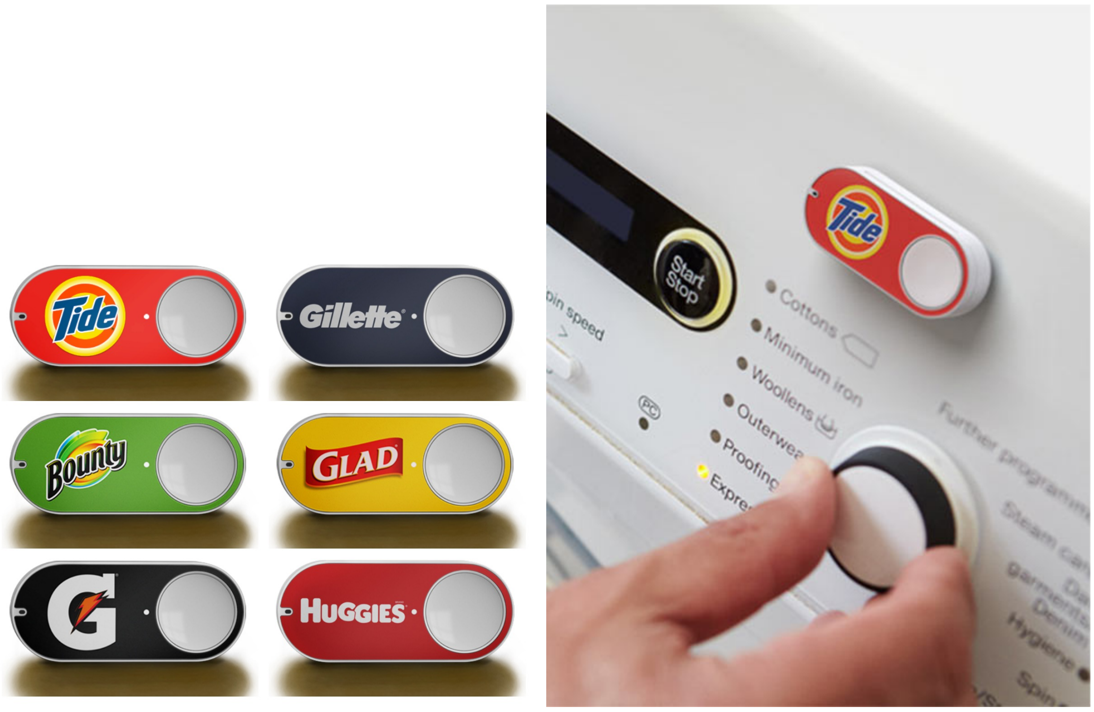


Figure 1: Amazon dash for each consumer product

image source : `https://www.amazon.com/Dash-Buttons/bie=UTF8&node=10667898011`
`http://wnep.com/2015/03/31/amazons-dash-button-lets-you-press-button-to-order-your-favorite-products/`
Accessed : 30/1/2017

in itself has a specific work group, IEEE-P2413 for IoT. A complete list of specification from IEEE working group for IoT is still underway. The working group focuses on defining the architectural framework domains of IoT. Also, it is responsible for building an abstraction and identification of the commonalities between the defined domains [25]. Similarly, IETF, a standardization body for IP and transport-layer protocols, has already rolled out solutions like DTLS [32] to facilitate end-to-end security in a heterogeneous IoT network. Also, there are many other proposals making their way to be a security solution for the IoT environment.

At the same time, researchers at academia have been doing extensive research for suitable security architectures and security protocols for the IoT environment. For example, Lian et al. [33] propose a security framework for multimedia devices. Sanaz et al. [34] have developed a secure and efficient Authentication and Authorization architecture for IoT-based health care. In these early days of IoT, based on existing and futuristic IoT device use case scenarios, several security protocols [26], [27], [29] are being developed and tested.

The aspect which differentiates IoT from conventional Internet is scalability. IoT is expected to host billions of connected devices. When the task is to configure and connect large numbers of devices, manual configuration of the devices will not be a convenient option. Especially, configuration involving complex steps will only make the user impatient and less interested in understanding the configuration. This in turn may leave many security loopholes. Instead, a transparent and user-friendly device configuration protocol will improve the chances of being aware of the configuration by avoiding pitfalls.

2.2 Secure Bootstrapping

The Oxford English dictionary describes bootstrapping as “the technique of starting with existing resource to create something more complex and effective” [30]. The principle remains the same in the context of both computers and network security. When a computer is started, an existing program named *bootloader* will bring the system to a fully functional state by loading and executing the necessary programs. Similarly, in the case of network security, when a device is connecting to the Internet, an existing program in the device will take care of establishing a secure connection from the device.

Several researchers [35], [36] have shared similar if not exactly the same definition for bootstrapping. There exists multiple ways to configure the security credentials (example, keying material or certificates) necessary for secure bootstrapping [48], [49], [35]. At the beginning of the life-cycle of a device, the security credentials can be embedded into the device while manufacturing. If not, a device vendor or a distributor can also install the credentials to the device. Alternatively, the necessary credentials can be derived by the device during boot strapping. However, compromise of any credential during the lifetime of a device will make the effect of bootstrapping and the overlaying security measures ineffective.

The importance of secure bootstrapping is still not widely understood. With the growing concern over security and privacy over Internet, secure bootstrapping can be a best practice for configuring a device. Now, in the case of a typical IoT device, after gaining network access, the device is expected to start data transaction with the back-end cloud service. The provision to maintain and monitor the IoT device from cloud service is expected to be part of the IoT framework. Additionally, the provision to gain access over the IoT device from remote locations can act adversely if suitable measures are not in place. Also, with billions of connected devices, the risk of security threats such as sniffing of user critical data and impersonation will be increased many fold. This makes secure bootstrapping essential than ever before. An ideal secure bootstrapping solution should finally grant access to Internet with inherent security features and with minimalistic interaction with the user.

2.3 Wi-Fi Networks

The application of Internet is expanding to multiple diverse fields like health monitoring, environment protection, road safety and intelligent agriculture [6]. A smart home is no longer a vision in the future but rather a fact in the present. Incrementally, the devices and appliances from our daily life are gaining Internet connectivity. This development is not confined to our home or work environment, rather, elaborate plans and architectures are getting proposed for smart cities and towns [7] to make our living most convenient. With all these developments underway, most convenient ways to join a network for the future devices needs to be analyzed.

Internet connectivity can either be wired or wireless, each being advantageous in its own way. However, the mobility feature offered by a wireless connection has been most useful. Many of the personal devices and appliances already have the facility for wireless connectivity. If not, the newer generation of same devices is at the verge of getting the wireless connection capability. Smart phone and notebook computers are the two most familiar electronic appliances with wireless connection capability. The common technologies used for wireless connections are cellular networks, Bluetooth, Zigbee and Wi-Fi. The digital cellular network in itself has three generation of technologies from GSM (Global System for Mobile communication) to LTE (Long Term Evolution). The infrastructure to support cellular networks is omnipresent. However, cost is major factor in using these technologies. This is because; a connection to any of these technologies requires subscription from a telephone operator. In an IoT scenario, subscription for each Internet connected device may not be desirable from the perspective of the user.

Bluetooth, Zigbee and Wi-Fi operate in global unlicensed frequency bands. However, they are still regulated by respective governing bodies and national authorities. Characteristically, three technologies differ in throughput and coverage area. Bluetooth is most suitable for personal area networks. At present, it is one of the effective ways for inter-device communication. The latest version of Bluetooth, Bluetooth-5, supports a coverage area of about 240 meters and a maximum throughput of 50 Mbps. Initially, Bluetooth was standardised as IEEE 802.15.1 but currently it is maintained by the Bluetooth Special Interest Group. Zigbee is also used for close range communication. It has a limited coverage of up to 100 meters and maximum throughput of 250 Kbps. Zigbee was standardised as IEEE 802.15.4 specification and is maintained by IEEE 802.15 working group. Both Zigbee and Bluetooth are energy efficient solution for wireless communication. Zigbee is mostly used in constrained devices. Whereas, Bluetooth can be used in both constrained devices and able appliances. In the IoT scenario, one topology for connecting Zigbee or Bluetooth capable devices to online services is by creating a personal-area network or an office mesh-network with an Internet-connected device. For example, with a smartphone as a hub and using the hub to connect other devices

Wi-Fi is a wireless technology for forming or joining a Wireless Local Area Network (WLAN) in the infrastructure mode. The Wi-Fi supported devices can connect to the Internet through an Access point (AP). Although connection to the Internet through Wi-Fi needs a subscription from an Internet Service Provider (ISP), a single ISP connection can connect one or more WLAN to the Internet. The major advantages of Wi-Fi over the Zigbee and Bluetooth can be listed as:

– Higher bandwidth: The latest specification 802.11ac allows 160Mhz of bandwidth. This enables to reach higher throughput up to gigabits per seconds.

– Greater coverage: Indoor coverage of about 100 metres and outdoor coverage of at least 300 meters are achievable [39].

– Easy to deploy and expand: Each AP can host at least a hundred devices. To host a larger number of devices, a network of APs needs to be installed.

– Cost effective: Infrastructure to host a Wi-Fi network already exists. A WLAN in infrastructure mode can be easily setup with off-the-shell devices.

– Direct connectivity : In an IoT scenario, devices as part of the WLAN will already have connectivity to the Internet.

Larger coverage and higher data rates are achieved in Wi-Fi by dissipating more power than Zigbee or Bluetooth technologies. However, recent advances made in defining new low power consuming wireless networking standard IEEE 802.11ah [43], also known as Wi-Fi HaLow, is defining several power efficient use cases for the Wi-Fi technology.

Nowadays, Wi-Fi networks are ubiquitous, familiar and convenient for users to connect and share information. Particularly in a home or office environment, users are already acquainted with Wi-Fi networks. Also, to fulfill the future spectrum demands on the cellular networks, new solutions are proposed for using the unlicensed frequency spectrum and co-exist with the Wi-Fi networks [5]. Adapting such solutions will make Wi-Fi networks an integral part of our daily life.

2.4 Security in Wi-Fi Networks

It is recognized that a WLAN is more prone to attacks than its wired counterpart. As the data is transmitted over the wireless channel, anyone in the coverage area of an AP has access to the data. A Wi-Fi connection between a device and an AP can either be of open type, with no security, or of secured type, with authentication and data protection. With the growing concern over user and data privacy, a secure connection for communication is either sought after or at least advised while using a public Wi-Fi connections. In an open connection, a user is particularly vulnerable to data the sniffing and security breaches [8].

A secure Wi-Fi connection ensures both authentication and data protection over the wireless link. The authentication confirms the identity of both the parties participating in data exchange, and invalidates the chances of an imposter to initiate or hijack a session. There exist two different methods of authenticating a station (STA). First, authenticate the STA at the target AP. Usually a pre-installed secret is used to validate the user at the AP. Second, authenticate the device with a back-end authentication server. In this case, the AP is connected to the authentication server with a secure connection and the AP only relays the protocol messages between a STA and the authentication server. This type of authentication is most common in corporate or industrial environment. The former method is more suitable for home environment.

Data protection splits down to two separate responsibilities, integrity protection and data confidentiality. Data at the receiver end should be able to guarantee that, over the air, it was not readable to anyone other than the involved parties, and the data was not modified during transmission. To ensure data integrity, cryptographically generated checksum is embed into every message. At the receiver end, similar checksum will be generated and compared with the received value. To guarantee the data confidentiality, both STA and AP will encrypt the message before the transmission and will decrypt the message after the reception. Keys necessary for data encryption, decryption and checksum generation are locally derived at both the ends of the connection from a shared master key set up during the authentication.

2.5 Security Protocols in Wi-Fi Security.

Security in a wireless network was taken into account even from the initial days of WLAN. As part of the first IEEE specification for WLAN [9], mechanisms for authentication and data privacy were already introduced. Since from the first specification to the latest one, WEP, WPA and WPA2 are the most prominent security protocols used in a Wi-Fi network.

2.5.1 WEP and WPA

As part of first IEEE specification [9], new data confidentiality protocol named Wired Equivalent Privacy (WEP) was introduced. As the name itself suggests, the goal of this new protocol, was to achieve privacy and confidentiality equivalent of a wired LAN in a wireless network. Soon after the release of WEP protocol, several attacks on the protocol were published [10]. Also, the protocol in itself had many design flaws [12]. The Wi-Fi alliance does not recommend the use of WEP, which is still supported in some legacy hardware. As a quick replacement for the flawed WEP protocol, Wi-Fi Protected Access (WPA) was introduced in 2003. WPA is also a security protocol for data confidentiality. However, the replaced protocol inherited some design flaws from its predecessor WEP and was a subject of several cryptographic attacks [13], [14] . Considering the limitations of the protocol, Wi-Fi alliance cautions users to not to buy hardware which only supports WPA.

2.5.2 WPA2

WPA2 is the security protocol which replaced WPA. It was introduced in 2004 and Wi-Fi alliance mandated the protocol for its certification in 2006. Any device which compiles to the Counter-Mode-CBC-MAC (CCMP) algorithm from IEEE 802.11i-2004 amendment, is certified as WPA2 compliant by the Wi-Fi alliance [15]. WPA2 uses the Advance Encryption Standard (AES) algorithm for data security and discontinues Rivest-Cipher-4 (RC4) based cipher systems. With this algorithm, a single 128 bit key is sufficient for ensuring data integrity and data confidentiality. However, adapting this new algorithm requires hardware changes.

2.6 Authentication in Wi-Fi Networks

Authentication is an effective method to implement access control in a network. In general, authentication can be done for devices or for users. At times, authentication of a device or a user can be used interchangeably. For example, every device belonging to a user may share the same credentials for authentication, and hence authenticating the device authenticates the user. In this thesis, authentication is done as part of device bootstrapping and it is always specific to a device. Authentication in the new EAP-NOOB protocol always refers to the device authentication.

There exists two different methods for authentication in both WPA and WPA2. The personal or PSK (Pre-Shared Key) method of authentication and 802.1X or commonly known as Wi-Fi Enterprise method of authentication. Methods differ in type of credentials used for authentication and the location of the registrar. The registrar is the entity which stores the credentials for the authentication.

2.6.1 WPA2-Personal

In WPA2-Personal, a passphrase is used for authentication. A passphrase is configured in the AP by the owner or administrator of the AP. Any associating STA should enter the same passphrase to successfully associate with the AP, and the passphrase is same for all the STAs. In WPA2-Personal, the registrar resides in the access point, so the authentication process ends in the AP. The encryption/decryption keys for the protocol will be independently derived at both STA and AP. The derived keys will be confirmed through a four-way handshake between the STA and AP.

However, there are few shortcomings with WPA2-Personal method. First, the security of the method completely depends on the strength of the passphrase. A weaker passphrase can easily be deciphered by a simple dictionary attack. Later, the encryption/decryption keys for every associated STA can be derived. Second, changing passphrase mandates a universal update. All the associated STAs should update the passphrase if the passphrase at the AP gets changed. Finally, the push button feature, which involves the Wi-Fi Protected Setup (WPS) protocol is vulnerable to attack [16] and compromises the security of the Wi-Fi network.

The WPA2-Personal is best suited for a small office or home environment. As there is no provision to individually manage the devices, it is better if this is deployed where there are only a few devices for the association. An effective level of security can be achieved by configuring a strong passphrase and turning off the WPS feature.

2.6.2 WPA/WPA2-Enterprise

This method offers a robust, flexible and easily manageable mode of authentication. A remote authentication server, which is connected to the AP with a secure connection, authenticates the associating devices. The registrar responsible for storing the credentials resides in the remote authentication server. The authenticating user or device will get assigned with a personal set of credentials like username and password. The Pair-wise Master Key (PMK), which is necessary to derive the session keys, is created in a cryptographic key-exchange during the authentication process. This method offers the flexibility to individually manage the device. Also, the method facilitates a centralized user or device management. In comparison with WPA2-Personal, this method demands skill and resources to install and maintain an authentication server. But in return, a more secure and flexible mode of authentication is offered. Thus, this mode of authentication is more suited for corporate and industrial environment.

Remote Authentication Dial-In User Service (RADIUS) is a networking protocol used for communicating between an AP and the Authentication, Authorization and Accounting (AAA) server. The RADIUS protocol operates from the application layer of a network stack. It functions in a client-server model. The client RADIUS application in an AP communicates with the server RADIUS application in the authentication server. The information about the RADIUS server should be configured manually in the AP. The secure connection between the client and the server RADIUS application is based on a pre-configured passphrase. The authentication servers can also form a hierarchy. The AP will only relay message to the lowest node in the hierarchy, and then the message will be relayed hop-by-hop to the correct RADIUS server. The credentials for authentication are on per-user or per device basis. Thus, during or after authentication, it is possible to exercise individual access control over a device.

The security features of the WPA2-Enterprise outweigh its counterpart. Further, WPA2 is more robust and secure than its predecessor. Also, considering that our goal is not only to provide a user friendly wireless device authentication protocol, but also to facilitate easy device configuration and management from a remote cloud platform. Hereafter WPA2-Enterprise will be the method of choice for authentication in EAP-NOOB.

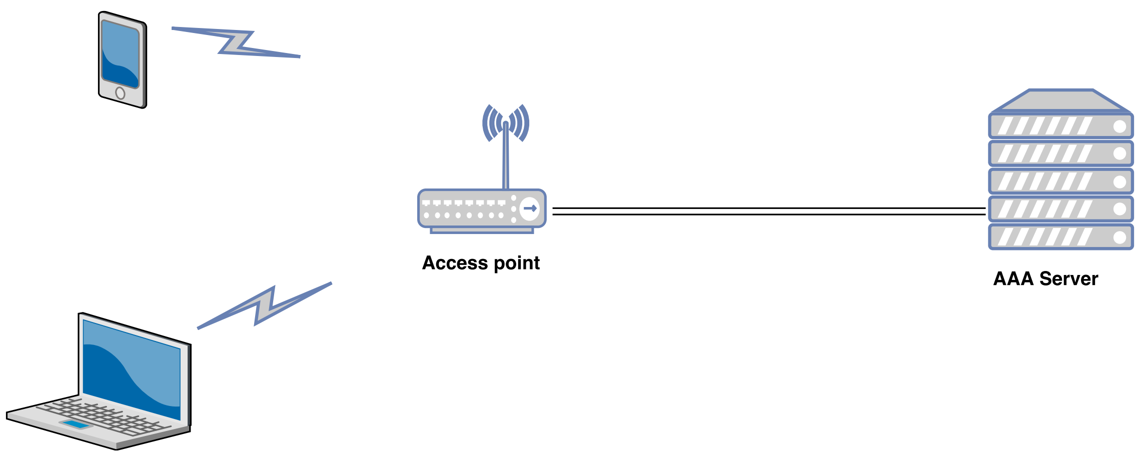


Figure 2: A WPA/WPA2-Enterprise network

2.6.3 IEEE 802.1X

IEEE 802.1X is a standard with authentication framework for enhancing the security in a LAN or a WLAN. In this standard, the user or device is authenticated by a central authority. However, the standard does not have its own method for authentication. Instead it makes use of EAP for authentication in both LAN and WLAN. This standard defines the rules to encapsulate the messages of the EAP framework in a LAN or WLAN.

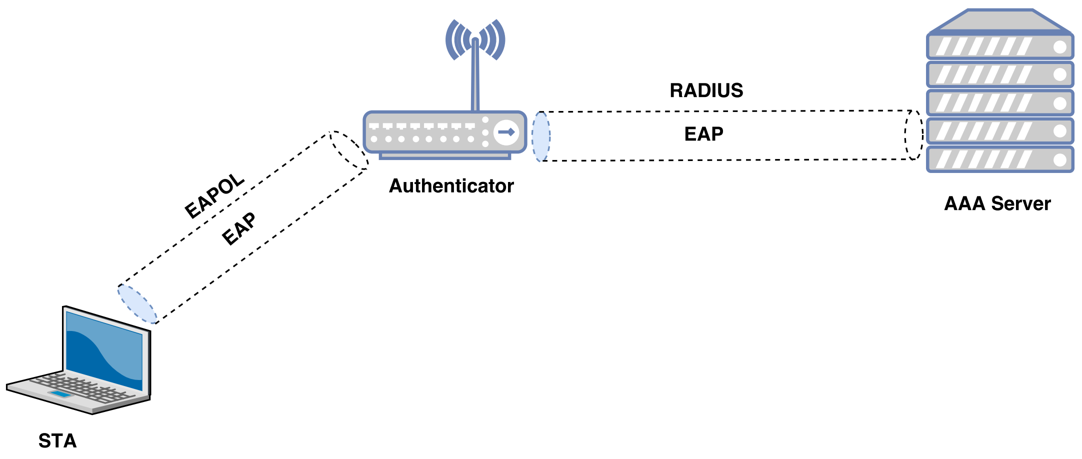


Figure 3: A IEEE 802.1X based connection

The terminology as per IEEE 802.1X is as follows: the authenticating device is called the supplicant, the device controlling the supplicant network access is the authenticator, and finally, the backend server responsible for the authentication is an authentication server. The standard has named the logical link between an AP and an authenticating device as EAPOL, which stands for EAP Over LAN. A typical network making use of IEEE 802.1X can be seen in Figure 3. RADIUS is the networking protocol used between the authenticator and authentication server. The successor for RADIUS is DIAMETER [44]. However, most of the existing infrastructure for establishing a WLAN only supports the older RADIUS protocol.

The authentication messages or specifically EAP messages are exchanged between the supplicant and authentication server. The messages are encapsulated as EAPOL between supplicant and authenticator. And between authenticator and AAA server, the messages are encapsulated as RADIUS messages. During the authentication process, the authenticator does not read or modify the EAP messages. Instead, it encapsulates the EAP message received from the supplicant in RADIUS message and decapsulate the message received from the authentication server. The authenticator is at the edge of the WLAN network. Hence, it is also an access controlling entity for supplicants in the WLAN.

2.6.4 Extensible Authentication Protocol (EAP)

EAP is an authentication framework which provides multiple methods for authentication. The EAP protocol operates at the data-link layer of the network stack. It only defines the message formats necessary for the authentication. It supports both wired and wireless networks, and is encapsulated within the frame format of the respective network type. For example in WLAN, EAP messages are encapsulated in a WLAN frame from a device supporting the IEEE 802.11 specification and encapsulated as per the IEEE 802.1X specification.

The terminology of the protocol is similar to IEEE 802.1X terminology. However in EAP, the supplicant can also be referred to as the peer. The strength of EAP is the flexibility that it offers. As per the concerned numbering authority, there are fifty five recognised EAP methods with a valid method identifier. Neither the authenticator nor authentication server has to pre-negotiate a method with a supplicant. Also, the authenticator need not understand the EAP methods. The authenticator can act as a pass-through authenticator [18] and it is how the authenticator is made use of in our EAP-NOOB protocol. When in a pass-through mode, implementation for any EAP method should only be implemented in the supplicant and the authentication server. This allows addition of new EAP methods into the EAP framework without making changes to the existing access network infrastructure.

After the establishment of EAPOL link between the supplicant and the authenticator, the first EAP message, Identity-request is sent from the authenticator to the supplicant. The message is to query the identity of the supplicant. The selected EAP method can sometimes be determined by parsing the identity attribute of the response Identity-response message. However, information about the selected EAP method will be exchanged in the subsequent EAP messages [18]. Both Identity-request and Identity-response are transmitted in plain text. The authenticator forwards the Identity-response directly to the authentication server. Then, the authentication server initiates the method specific EAP message exchange. A series of messages are exchanged between the server and the supplicant before concluding the message exchange. The message exchanging session should always conclude either with EAP-SUCCESS or EAP-FAILURE. All the messages between an Identity-request till EAP-SUCCESS/FAILURE belong to a single EAP session.

In all the sessions which end with an EAP-SUCCESS, the master session key MSK will be derived at both the supplicant and authentication server. The MSK is derived using the message parameters from the recent EAP session, and possibly with some method-specific inputs. Using MSK, the Pairwise Master Key (PMK) will be derived at supplicant and authentication server. Then, the authentication server will export the PMK to the authenticator. Once the authenticator receives the PMK, the next task for the authenticator and supplicant is to mutually confirm that the PMK at both ends are identical. This is done with the help of a four-way handshake procedure [19]. During the handshake procedure, both sides derive pair-wise transient key (PTK), a session key which will be divided into five separate transient keys. The transient keys from PTK will be used for securing the user identity and data over the wireless network.

As part of the handshake procedure, both the authenticator and supplicant will generate and exchange a message integrity check (MIC) digest. The MIC digest is computed using one of the transient key derived at both the ends. Recreating the MIC digest, same as the received one will confirm that the supplicant and authenticator are having identical PTK and PMK. The lifetime of the PMK will be greater than PTK. Also, during the lifetime of PMK, multiple PTK can be derived. The four way handshake confirms that the PMKs are identical without disclosing the keys. This is critical for the secrecy of the key. Four-way handshakes were introduced as part of IEEE 802.11i-2004 amendment and is extensively used to generate or renew the keys in a wireless network.

2.7 Secure Device Configuration.

In an environment with multiple IoT devices, the owner or administrator configures the device to use the functionality of the device. This requires understanding of the common technologies involved in the configuration of a device. Misconfiguring a wireless network may create security vulnerabilities to be exploited and may affect every member of the network [17]. Ideally, each device should have its own set of credentials to ensure the security of the devices. In this section, we will discuss the protocols and concepts necessary for secure configuration of a device in a WLAN.

2.7.1 PKI and Digital Certificates

Uniquely identifying an entity and validating the received identity are the most important steps of the authentication process. Digital certificates, username/password, challenge-based password and digital tokens are some of the most commonly used credentials in EAP methods for authentication. The level of security for a selected type of credential depends on various factors. For example, the strength of a password is a major factor in password based authentication. However, authentication based on a digital certificate is considered to be the most secure method [20].

Digital certificate is an electronic document which has the inherent characteristics to prove the authenticity of the owner of the certificate. In the real world, it can be compared with the passport issued by any sovereign country. Digital certificates are issued by a trusted third party known as the certificate authority (CA). The CAs issues signed certificate to the requesting entity. The signature from the CA legitimizes the issued certificate. The attribute named as Subject in a certificate holds the name of the owner for the certificate. Other attributes of a certificate include the version, serial number, validity, public key of the owner, algorithm used for public key, signature from the CA and the algorithm used for signature [21]. Apart from the user or device authentication, digital certificates also can provide non repudiation, high level of information confidentiality and strong data encryption [22].

Digital certificates are part of a large system called public key infrastructure (PKI). The PKI defines the policies and rules necessary to create, distribute and manage digital certificates. PKI contains four specific authoritative bodies [23]:

- Certificate authority(CA): Is the root of trust, responsible for issuing digital certificates.
- Registration authority(RA): A subordinate to CA. It verifies the user requests and recommends CA about issuing the digital certificate.
- Repository : Stores and distributes issued certificates. This could be a single entity or a collection of distributed entities. Additionally, the repositories are not commonly part of Internet PKI.
- Certificate revocation list (CRL) issuer: An optional subordinate for CA which has the authority to issue revocation list for the issued certificates.

In the EAP framework, EAP-TLS, EAP-TTLS and PEAP are the most prominent EAP methods which make use of digital certificates for authentication. Apart from the EAP framework, digital certificates are extensively used in Internet banking, e-commerce and secure E-mail. A user can certainly generate a certificate and sign it. However, self-signed certificates are not very useful, as they cannot be trusted. A popular practice is to buy certificates from trusted commercial CAs, such as Verisign[2]. In an IoT environment, the devices which are part of an ecosystem may own certificates provided by the device manufacturer or vendor. If the device does not contain a certificate or the CAs of the installed certificates are unknown, then the user has to buy and install new certificates to each device. Given the long lifetime of a device, the need for replacing the certificate in its lifetime cannot be ruled out.

2.7.2 Diffie-Hellman Key Exchange

Some of the key components in this thesis rely on the Diffie-Hellman key exchange. In this section we briefly describe the key exchange procedure. Establishing a secure communication on a public network is not a trivial task. Consider two unknown entities that want to have a secure communication over an unprotected network. To ensure the security, the communicating entities can encrypt their messages and be rest assured about the secrecy of the messages over the network. The Message encryption and decryption will require shared keys. They can be delivered in two different ways. First, a trusted independent entity can distribute the keys to both communicating parties over a secure channel. The independent entity can usually be used to distribute keys, as it is either too slow or expensive to send user messages through it. The second method is to derive the same set of keys, independently, at both the ends. Before the key derivation, both the entities will generate a public and private key pair. The public keys can be exchanged between the entities in plain text. Later, cryptographic operations are performed to derive shared secret keys. For the key derivation, each endpoint's own private key and the received public key are used as input parameters. This method of key distribution is generally known as public-key cryptography.

A detailed explanation of the Diffie-Hellman key distribution is as follows,

Consider the communicating entities as A and B.

- Two prime numbers g and p are selected by one of the entity and communicated to another in plaintext.
- A selects a random number x and computes $m = g^x mod\ p$ and sends m to B.
- B selects y, a random number, and computes $n = g^y mod\ p$. Then, n is transferred to A.
- A derives the shared key by operation $K = n^x mod\ p$. Similarly, B derives the same key by operation $K = m^y mod\ p$.
- In the end, both A and B will have the shared key $K = g^{xy} mod\ p$.

[2]http://www.verisign.com/

An eavesdropper can obtain g, p, m and n. However, obtaining x from m will require discrete logarithm operation on m.

$x = log_g\ m\ mod\ p$

Similarly,

$y = log_g\ n\ mod\ p$

Obtaining x from m and y from n can be difficult for carefully chosen values of p and g. It requires operations in the order of $p^{1/2}$ to compute either x or y. Only an algorithm whose complexity grew at a rate of $log_2\ p$ can break Diffie-Hellman system . The security of this technique relies on the difficulty of computing the logarithm of $mod\ p$ [37]. The logjam attack [38] made use of a flaw in TLS (Transport Layer Security), which internally used Diffie-Hellman for key distribution. In it, the TLS connection is downgraded to make use of a 512-bit p and eventually compromised the security of the connection. Additionally, the possibility to break the protocol with a 1024-bit p was also discussed. To avoid the attack, it was advised either to make use of a 2048-bit prime number for p or use the elliptic curve variant of Diffie-Hellman, which does not require larger values of p.

The elliptic curve variant of Diffie-Hellman is used for key distribution in our EAP-NOOB protocol. The decision is based on the fact that shorter keys can be used which is less computation intensive.

2.7.3 Elliptic Curve Diffie-Hellman (ECDH)

Here, the public key cryptography is applied on a finite field elliptic curve. As in the original Diffie-Hellman scheme, the security of the elliptic curve variant relies on the infeasibility to solve the discrete logarithm problem. Here it is referred to as the elliptic-curve discrete logarithm problem. The shared secret is calculated by performing finite set operations on the curve, precisely the point multiplication of a coordinate on the curve with an integer.

Before the start of key distribution, the involved entities agree on the system parameter: the finite field p, curve constants a and b, and the base point G. It can also be done by agreeing to a standard set of parameters defined for a curve by NIST. Now, consider the communicating entities as A and B. The steps to derive shared key between A and B are as follows,

– At first, the system parameters are shared or agreed on.

– A selects a random integer x and computes $m = G^x$. Here x is the private key and m is the public key. Public key is sent to B.

– Similarly, B selects a random integer y and computes $n = Gy$. Then n is sent to A.

– Shared secret $K = n^x$ is computed at A and $K = m^y$ at B.

– Both A and B have finally derived $K = G^{xy}$.

ECDH is only a key agreement protocol and does not authenticate the identity of the participating entities. This has to be handled by external methods. Authenticating the entities will avoid impersonation and man-in-the-middle attacks. The public keys in the key agreement can be either static or ephemeral. Static public keys are undesirable as it reduces the random inputs to the protocol. Moreover, the use of ephemeral public keys ensures perfect forward secrecy in the protocol.

When compared to popular public key cryptosystems, the Elliptic Curve Cryptography (ECC) needs shorter keys to provide the same level of security. For example, an elliptic curve over a 163 bit field is as effective as a 1024 bit modulo p RSA or Diffie-Helmann. The security increases with larger keys so that the security offered by a 571 bit field is equivalent to 15,360 bit RSA/ Diffie-Helmann [11]. The shorter key will be less computation intensive, and hence ECC are especially useful for constrained wireless devices.

2.8 Out-of-band Channel for Authentication

A logically and physically independent communication channel, other than the existing channel for data transfer between two communicating entities, can be termed as an out-of-band (OOB) channel. In research literature, there exist many synonyms for an OOB channel. They include auxiliary channel, human assisted channel and manual channel [36]. Conventionally, the OOB channel is used for notifying a state or an event which cannot be sent over the in-band channel. Now, in the context of network security, the OOB channel has been extensively used for authentication. This is because the OOB channel often is robust against attacks. According to Shahab et al. [36], the security properties of an OOB channels may include, proven identity based authentication, authenticity of the origin of data, data integrity and confidentiality.

The in-band communication is secured by completing the authentication procedure using the OOB communication. For the user, in comparison with authentication over PKI infrastructure, there will be a slight increase in the complexity of the authentication procedure. Also, when the OOB based authentication is used as the secondary verification method for an existing method (two factor authentication), it increases the credibility of authentication and in turn enhances the security. When used as the only method for authentication (single factor authentication), OOB based authentication still can provide strong and reliable authentication.

Based on their nature, OOB channels can be classified into three different categories: private, public and weak [36]. In a private OOB channel, the confidentiality of the data is of utmost importance. Special precautions are taken to avoid disclosure of contents of the message while passing it over the OOB channel. An example of a private channel is a secret string read by the user from one device and typed manually into another device. Additionally, it must be made sure that the secret from first device is only read by the user and nobody else. In a public channel, the message to be passed over the OOB channel is publicly available. An example of a public channel is scanning a barcode from a public display. A message on the public channel can be sniffed, blocked and delayed. Finally, a weak OOB channel contains all the characteristics of a public channel, but additionally, the message can be stored and replayed. An example of weak channel is passing a voice message between two devices over wireless interface. Both public and weak channel can only ensure integrity and authenticity of data. All three categories of OOB channel may exist on the same physical medium. However, the channels are classified based on the security feature adhered by the medium while transferring OOB messages between the two entities.

An OOB channel can introduce flexibility in an authentication process. For example existing infrastructure like cellular and Wi-Fi networks can be used for OOB message passing with the help of familiar user equipments such as smartphones and smart phones. In the case of constrained devices, there may not be provisions to host complex security infrastructure for authentication. In such cases, the OOB channel based authentication can act as a alternative. The OOB channel authentication prevents difficult to conduct security attacks such as the Man-in-the-middle attack.

The EAP-NOOB protocol makes use of a user-assisted OOB channel. In this protocol, the message which is over the OOB channel contains a secret nonce and a cryptographic hash for integrity protection. Although the cryptographic hash will confirm the integrity of the message at the receiver's end, it is recommended not to revel the secret nonce over the transmitted OOB channel. Thus a private OOB channel will be most suitable for EAP-NOOB.

3 Protocol Description

In this chapter we provide a detail description about the EAP-NOOB protocol. We start the chapter with an overview of the protocol. Next, we describe the terminology of the protocol and the protocol state machine. Later in the chapter, we provide a detailed description about each stage of the protocol. Finally the chapter ends by discussing the error handling and the security offerings of the protocol.

3.1 Overview

EAP-NOOB is a secure bootstrapping protocol that relies on a user-assisted OOB channel. It is added as a new authentication method to the EAP framework. The successful authentication requires message exchange over multiple EAP sessions. The protocol is responsible for key derivation, authentication and registration of IoT devices. The protocol is for off-the-shelf devices with limited user interface capabilities. A display unit or a camera are some example devices that could use the EAP-NOOB protocol. Devices with limited or no user interfacing capability that cannot send and receive dynamically generated messages are not the target devices for this protocol. The protocol is equally effective over devices with or without pre-registration, pre-installed credentials and pre-provisioned device identifiers. At the end of the bootstrapping procedure, the device will be registered with an authentication serve, the ownership of the device is established, and the credentials necessary for secure network access and application-level security has been derived.

For the successful completion of bootstrapping, an OOB message exchange between the device and the authentication server is mandatory. As part of the bootstrapping, Diffie-Hellman key exchange is performed over an open channel. The message from the OOB channel establishes the ownership, authenticates the key exchange and completes the shared key derivation. It is difficult to launch a Man-in-the-middle attack on the protocol, as it requires contents of messages from two independent channel to complete the attack. Also, the OOB message is dynamically generated. Compared to static messages, the dynamic code is more secure against dictionary and social engineering attacks. The OOB channel is expected to be partially automated. This grants the freedom to keep the message size in the order of tens of bytes. Also, it avoids manual operations such as, memorising and typing a message. An example OOB message passing scenario is scanning a QR code from a display unit through a smart phone.

The EAP-NOOB is specified as an open standard and a protocol for secure bootstrapping of IoT devices. The protocol is considered to be generic as it does not rely on any single OOB channel. Not depending on pre-installed credentials and identifiers enables easy transfer of device ownership. Also, the ability to support different types of user interfaces makes the protocol applicable to heterogeneous environment.

3.2 Protocol Terminology

The following are the important terminology that are used to explain the protocol.

- authenticator: Common entity in both EAPOL and RADIUS links. In this protocol, the entity functions in pass-through mode
- peer/client/supplicant: The IoT device under bootstrapping.
- authentication server/AAA/EAP server : EAP session terminating entity. It is also referred to as server in this section.

3.3 State Machine

The EAP-NOOB is a manifestation under client-server modeled EAP protocol. A peer will contain client side EAP-NOOB and the authentication server will have the server side EAP NOOB. The protocol consists of five distinctive states, and follows event based state transition. The states are named as: *Unregistered(0), Waiting for OOB(1), OOB received(2), Registered(4) and Reconnecting(3).* Both peer and server have similar state machines. Also, the state changing event affects alike on both the state machines. The state machine and state transitions can be seen in Figure 4.

A server side EAP-NOOB can interact with multiple peers and it maintains a state machine for each peer. Similarly, a peer may interact with multiple authentication servers initially. However, as the bootstrapping is completed with only one of those server, the peer side EAP-NOOB will only have a single state machine transitioning till *Registered* state. At the beginning of bootstrapping, both peer and server should be at *Unregistered* state. Also, in the event of loss of persistent information, the state machines should be reset to *Unregistered* state. Always, the peer will initiate the EAP-NOOB method. Later, based on the received EAP-NOOB message at the server and the current state of both peer and server, the server will decide the next message and next state transition.

Once at *Unregistered* state, the peer can do Initial exchange with multiple servers. Initial exchange is a parameter negotiation session for identifying the suitable server. Completion of Initial exchange will change the state from *Unregistered* to *Waiting for OOB* state. Waiting exchange, a probing session for an OOB message is done at *Waiting for OOB* state and it will not result in a state transition. Initial exchange and Waiting exchange will always end with an EAP-FAILURE, terminating the EAP session. Then, based on the negotiated OOB message direction, the OOB message sender (OOB output sender) will remain in the *Waiting for OOB* state. However, the receiver of the OOB message (OOB input receiver) will transit to *OOB received* state. The states *Waiting for OOB* and *OOB received* are considered ephemeral. Initial exchange in the OOB received state will change the state to *Waiting for OOB* state. However, Initial exchange in *Waiting for OOB* state will not result in a state

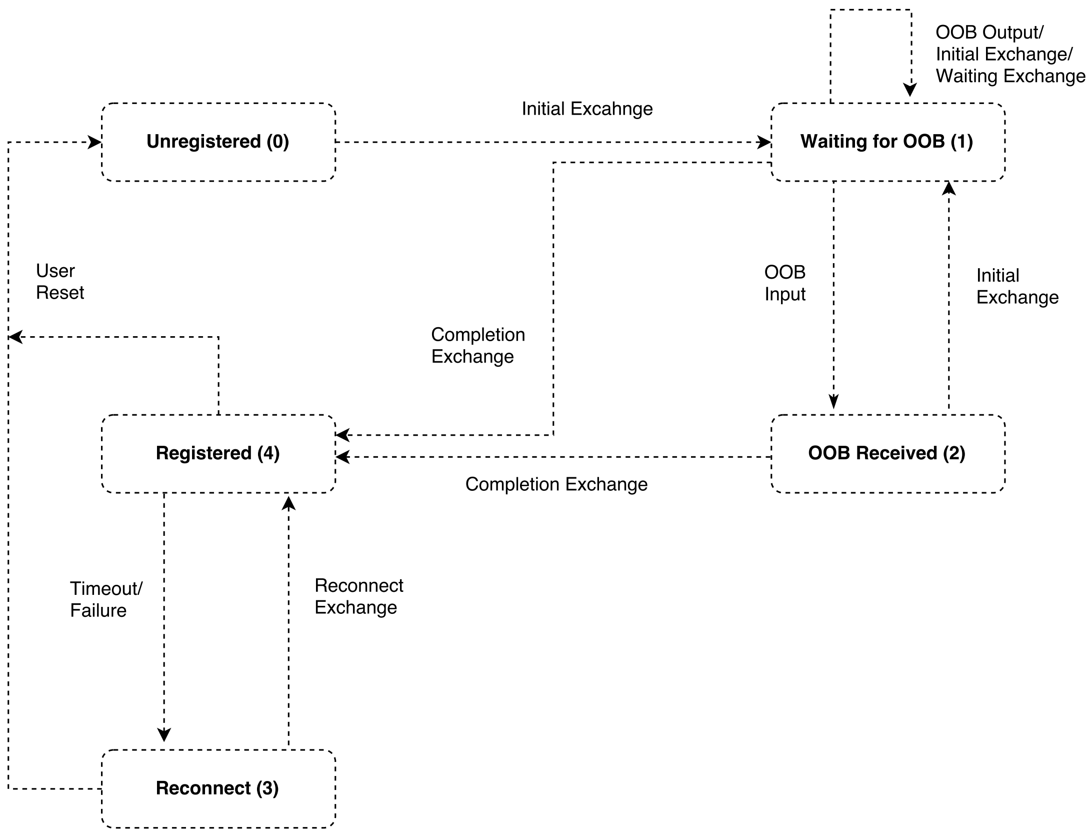


Figure 4: EAP-NOOB state machine

transition. Although a peer can do Initial exchange with multiple servers, the OOB message is sent to or received from a single server with which the user has an account.

An OOB message exchange between the server and peer should only happen once for each peer. At the end of a successful Completion exchange, a shared secret derivation and confirmation procedure, the OOB output sender will directly transit from *Waiting for OOB* state to *Registered* state. Next, the OOB message receiver will transit from *OOB received* state to *Registered* state. Later, the state machines will toggle between *Registered* and *Reconnect* state, unless the device is reset or either of the entities losses the persistent information. Falling back from the *Registered* state on either side should only happen with user intervention.

It requires a minimum of two EAP sessions to successfully complete the EAP-NOOB method. Also, the protocol requires mandatory user assistance to successfully complete the authentication. The total time and the complexity of user assistance depend on the chosen OOB channel. Therefore, one cannot pre-estimate the total authentication time. Although, an EAP session does provide timer based waiting mechanism for receiving a message. In case of EAP-NOOB, a fixed time bound approach to complete the user assistance is not a feasible solution. Hence, the method is completed using multiple EAP sessions. Also, it helps to segregate the method into three distinctive steps: peer interaction with all the relevant servers through Initial exchange, Waiting for OOB message through Waiting exchange and finally, shared key confirmation through Completion exchange. After successful registration, every re-keying is done in a separate EAP session.

3.4 Protocol Stages

In this section of the thesis, we describe the messages and their contents for all the stages of EAP-NOOB protocol. Every EAP-NOOB message will bear a message identifier (ID) as part of the message. The identifiers span from 0 to 7. Message ID 0 is reserved for error messages and rest are used at different stages of the protocol. Characteristics of each of the message parameter are covered in Table 6 of Appendix A.

3.4.1 Initial exchange

The EAP-NOOB supporting device will initiate the bootstrapping procedure with the Initial exchange. It is started by responding to the EAP-Request/Identity message from an authenticator. The server need not have to have any prior information about the peer. Instead, the peer will use a generic network access identifier (NAI) string *noob@eap-noob.net* in its reply with every server. The receiving server will allocate a unique identifier, named peerID, for each interacting peer. The peerID is used to uniquely identify the client for the complete duration of association and it is also helpful for maintaining the peer context in the server. In case of Initial exchange from any ephemeral state, the NAI will be specific and will have the peerID and the current state information of the peer. The NAI follows the format *peerID+sX@eap-noob.net.* Here, s is a character delimiter for state and X refers to current peer state. In every Initial exchange, the reply to EAP-Request/Identity will only contain the current NAI of peer.

After exchanging identities, negotiation of parameters will take place. The peer and server parameter sets containing: protocol version, OOB message direction and supported cryptographic algorithms are exchanged over Type 1 EAP-NOOB message. Type 1 EAP-NOOB Request from server will contain information about supported cryptographic algorithm (Cryptosuites), supported OOB message direction (Dirs) and Supported EAP-NOOB protocol versions (Vers). The peer will then match the received parameters with its own and will reply with a Type 1 EAP-NOOB-Response message. The reply message contains selected cryptographic algorithm (Cryptosuitep), supported version (Verp) and OOB message direction (Dirp). Also, specific set of information about the server and the peer are exchanged as parameter serverInfo and peerInfo. The values from these parameters are essential to identify or access either a server or a peer from the application layer. When the selected OOB message direction is peer-to-server, a user will select the relevant server based on the serverInfo. Similarly, when the selected direction is server-to-peer, a user will look for the device information at server side based on peerInfo.

Next, nonces (Ns and Np) and ECDH Public keys (Pks and Pkp) from both server and peer are exchanged over the Type 2 EAP-NOOB messages. The exchanged public components should be based on the negotiated cryptosuite and are later used for deriving the shared key. Finally, the server will end the session by sending a EAP-FAILURE. Initial exchange with its parameters as represented in [42] can be seen in Figure 5.

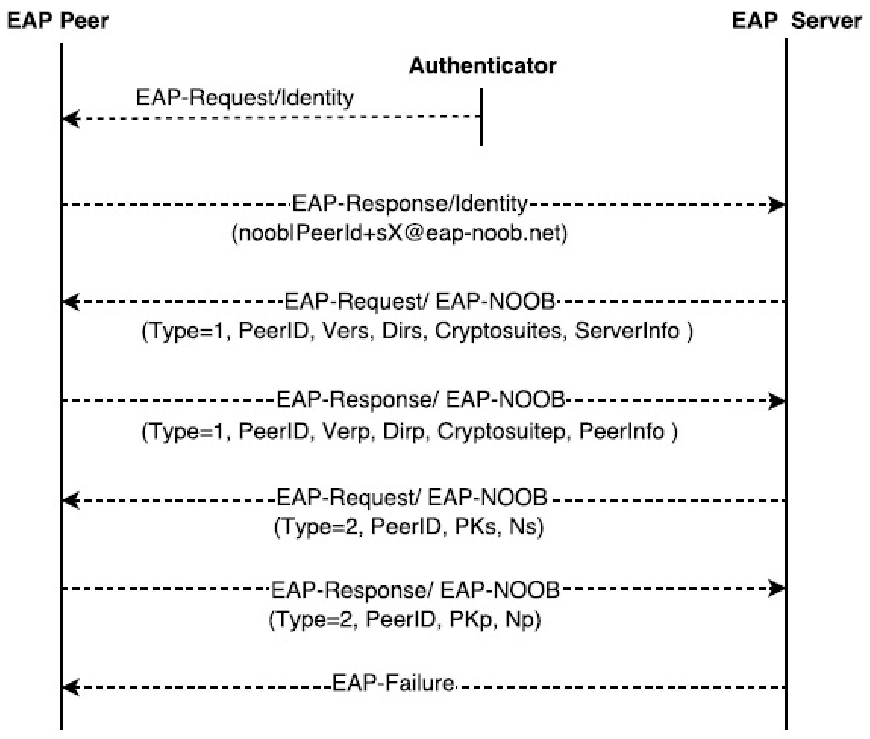

EAP Peer
EAP Server
Authenticator
EAP-Request/Identity
EAP-Response/Identity
(nooblPeerId+sX@eap-noob.net)
EAP-Request/ EAP-NOOB
(Type=1, PeerID, Vers, Dirs, Cryptosuites, ServerInfo)
EAP-Response/ EAP-NOOB
(Type=1, PeerID, Verp, Dirp, Cryptosuitep, PeerInfo)
EAP-Request/ EAP-NOOB
(Type=2, PeerID, PKs, Ns)
EAP-Response/ EAP-NOOB
(Type=2, PeerID, PKp, Np)
EAP-Failure

Figure 5: Initial-exchange

3.4.2 OOB Step

Based on the negotiated message direction in the Initial exchange, an OOB message is sent either to the peer or the server. When the negotiated direction is peer-to-server, based on the OOB channel and the peer side user interface capability, a peer may generate more than one OOB message, i.e one OOB message for every completed Initial exchange. Then, the user will select the relevant server by selecting the corresponding OOB message generated for that particular server. However, when the negotiated direction is server-to-peer, there will only be one OOB generated.

This is because the server can directly look up for the relevant peer context in its own database and can generate the OOB message for the retrieved peer context. As the OOB message is transferred over an user assisted channel, the OOB message will not contain a message type. Instead, the OOB message will always have three parameters: server assigned peer identifier (peerID), an OOB message nonce (Noob) and a cryptographic finger print (Hoob). The Hoob is generated using the parameters from Initial exchange.

(Hoob = H(Dir, Vers, Verp,peerId,Cryptosuites,Dirs,serverInfo, Cryptosuitepp,Dirp,peerInfo,PKs,Ns,PKp,Np,Noob).

The cryptographic hash generating function *H* is selected based on the negotiated cryptographic algorithm from Initial exchange. The input parameter *Dir* used for Hoob generation is the negotiated OOB message direction. When the negotiated direction is three, both OOB input and OOB output should be delivered. At the receiver's end, the integrity of the received OOB message should be verified by computing the Hoob using same inputs, and comparing it with the received value.

3.4.3 Waiting exchange

Although an OOB message is sent over an OOB channel, confirmation of reception of OOB message over the in-band-channel is necessary. A server will have to service multiple peers during its up-time, and continuous probing for the confirmation of reception will utilize the server resources unnecessarily. Instead, a Waiting exchange will be done. Irrespective of the OOB message direction, a Waiting exchange will always originate from a peer. Each Waiting exchange is done over a separate EAP session. Right after the Initial exchange, when the peer and server are at *Waiting for OOB* state, the peer will respond to EAP-Request/Identity with its current NAI. The server then assigns a wait time in seconds to the peer. Till the end of the assigned time, a peer should refrain itself from probing the server. The server will assign the wait time over a Type 3 EAP-NOOB-request. To which, the peer will respond with a Type 3 EAP-NOOB-response containing the peerID as a acknowledgment. Finally, the server will end the session with an EAP-FAILURE. A Waiting exchange message sequence as represented in EAP-NOOB specification [42] can be seen in Figure 6.

When the OOB message is received either at a server or a peer, the next probing EAP-Response/Identity will lead to Completion exchange. The automatic transition from *Waiting for OOB* state to *OOB Received* state itself is the confirmation of reception of an OOB message. Waiting exchange also helps to set an upper time limit for the number of OOB message exchanges. When a peer completes Waiting exchange for a certain number of times without an OOB message, then the server will send *Unwanted peer* error message to the peer. After receiving the error, the peer should refrain from probing for a long interval. As it is the responsibility of the peer to probe and confirm the reception of an OOB message, Waiting exchange in

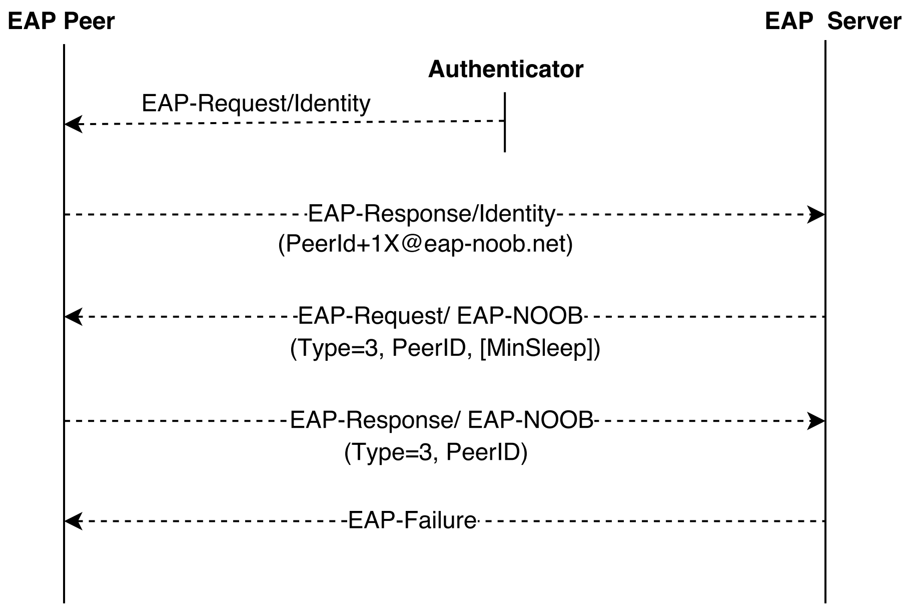


Figure 6: Waiting-exchange

server-to-peer direction may not seem necessary. However, with the help of Waiting exchange, the concept of setting a maximum time duration for OOB step can be used in server-to-peer direction. So the procedure is carried out in both the directions.

3.4.4 Key Derivation for Completion exchange

After receiving the OOB message, both server and peer will derive the shared key Z based on the ECDH algorithm. The Ephemeral Unified Model or C(2,0,ECC,CDH) scheme from NIST specification [40] is used for key derivation. Under this scheme, two ephemeral keys and two static keys are derived. The hash function H used for concatenation key derivation function (KDF) is from the negotiated cryptographic algorithm. The input parameters for the KDF are: shared key between the entities, eight byte ASCII string "EAP-NOOB" as AlgorithmID, Np as PartyUInfo, Ns as PartyVInfo and Noob as SuppPrivInfo. The parameter SuppPubInfo from NIST specification is not provided for key derivation during Completion exchange. The input parameters: Np, Ns and Noob should be 16-byte byte strings.

The derivation will yield 192 byte shared key Z. Later, the shared key is divided into five separate sub keys. The sub keys are MSK (byte 0-63), EMSK (byte 64-127), KMs (byte 128-143), KMp (byte 144-159) and Kz (byte 160-191). MSK and EMSK are used by EAP as keying materials for deriving transient keys. KMs, Kmp and Kz are used internally by EAP-NOOB for computing message authentication codes (HMAC).

3.4.5 Completion exchange

After the reception of an OOB message, either server or peer will first reach the OOB received state. The next EAP-Identity/Response from peer will trigger the Completion exchange from server. The server will prepare a Type 4 message containing server message authentication code (MACs) and will send it to the peer.

MACs = HMAC(Kms; 2,Vers,Verp,peerId,Cryptosuites,Dirs,serverInfo, Cryptosuitep,Dirp,peerInfo,PKs,Ns,PKp,Np,Noob).

The parameters from the Initial exchange, OOB message nonce (NOOB), server part of the shared Key (KMs) and a character '2' are provided to the HMAC function to generate the MACs. After receiving Type 4 EAP-NOOB request, the peer will also generate MACs with the same set of inputs and will compare it with the received value. The peer will respond with an error message if the comparison for MACs fails, else a Type 4 EAP-NOOB response with the peer message authentication code (MACp) will be sent.

MACp = HMAC(Kmp; 1,Vers,Verp,peerId,Cryptosuites,Dirs,serverInfo, Cryptosuitep,Dirp,peerInfo,PKs,Ns,PKp,Np,Noob).

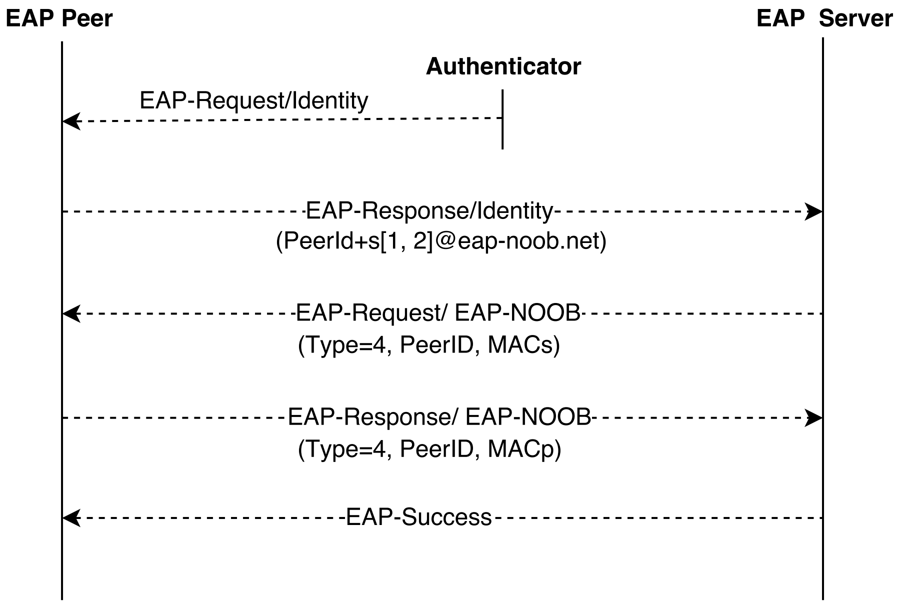


Figure 7: Completion-exchange

The HMAC function at peer will use parameters from Initial exchange, OOB message nonce, peer part of the shared key (KMp) and character '1' for computing MACp. After receiving the MACp over Type 4 EAP-NOOB response, the server will compute its own MACp with the same set of input parameters and will compare it with the received value. An error message will be sent if MACp comparison fails, else, the server will respond with an EAP-SUCCESS, which will also end the EAP session. The EAP-SUCCESS message confirms the derivation of same shared secret at both the ends. Later, the MSK and EMSK part of the shared secret are exported from server to the authenticator. Then, the authenticator will perform a four-way handshake with peer to derive the transient keys. The keys are used for data confidentiality and integrity protection over the wireless network. Successful completion of an EAP session will grant network access permission to the client device (EAP peer). At the end of a Completion exchange, both peer and server will move to the *Registered* state.

3.4.6 Reconnect exchange

If the PMK at AP expires, or the Master key at either entity is lost either due to a hardware or a software failure, a Reconnect exchange is performed. Upon discovery of either of the events, the state machines at both entities will transit to *Reconnecting* state. After the state transition, the next EAP-Identity/Request will be responded with the current NAI of the peer. This will trigger the Reconnect exchange. When in *Registered* state, the peer should never initiate the Reconnect exchange. The Reconnect exchange involves fast reconnect feature from the EAP specification [18]. An important thing is, the OOB message exchange is not necessary for the Reconnect exchange.

In Reconnect exchange, three pairs of EAP-NOOB messages are exchanged. The procedure will start by exchanging the Type 5 EAP-NOOB Request/Response messages. Wherein, the peer and the server will once again negotiate the cryptosuite. Optionally, the peerInfo and serverInfo parameters can also be included in the Type 5 Request/Response message. Next, in the Type 6 EAP-NOOB Request/Response message, fresh nonces (Ns2 and Np2) and ECDH public keys (PKs2 and PKp2) are exchanged. If the negotiated cryptosuite is same as the cryptosuite from the Initial exchange, then the public key components (PKs and PKp) are not exchanged.

Similar to Completion exchange procedure, the server and peer will exchange message authentication codes (MACs2 and MACp2) for confirming the derivation of same shared secret. The message authentication codes are computed using the newly derived server and peer part of the shared keys (Kms2 and Kmp2). The codes are exchanged over the Type 7 EAP-NOOB Request/Response. Successful comparison of message authentication codes will confirm the derivation of same shared key at both the entities. The server will then terminate the EAP session with an EAP-SUCCESS message. Also, the newly derived MSK and EMSK are exported from the server to the authenticator. This will initiate a four-way handshake between the authenticator and the peer. After successful Reconnect exchange, both server and peer will move back to the *Registered* state. In case of mismatched MACs2 or MACp2, an error message will be sent from the message receiver to the sender, followed by an EAP-FAILURE from the server. A Reconnect exchange message

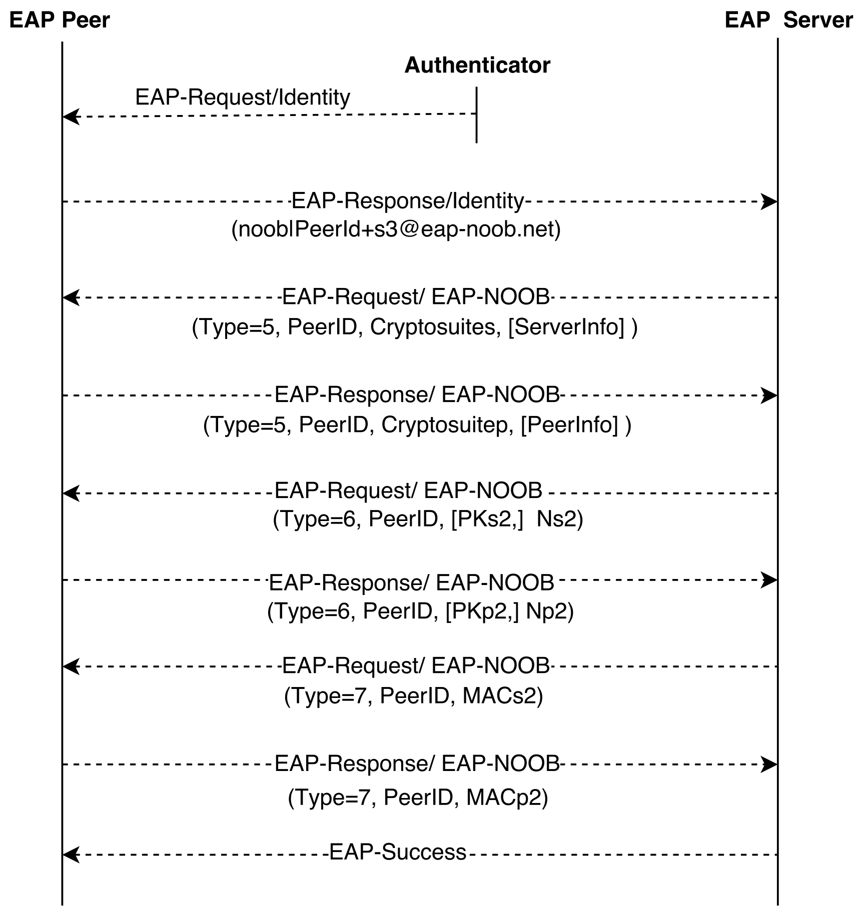


Figure 8: Reconnect-exchange

sequence as represented in EAP-NOOB specification [42] can be seen in Figure 8.

3.4.7 Key Derivation for Reconnect exchange

Based on the parameters exchanged as part of Type 5 and Type 6 EAP-NOOB messages, a shared key can be derived using two separate methods. In the first method, if the negotiated cryptosuite in the Type 5 message is different from the existing cryptosuite, then a new shared key is derived. It is done using the hash function from the concatenation key derivation function of the new cryptosuite. The KDF output Z is derived with a new MSK (byte 0-63) and EMSK (byte 64-127). The sub key Kms2 and Kmp2 are 32 byte long and are formed by concatenating the values from the newly derived Z (byte 128-143 and byte 144-159) and the stored Kms and Kmp. Finally, the newly derived Kz (byte 160-191) will directly replace the old one.

The second method is only applicable when the negotiated cryptosuite is same as the cryptosuite from Initial exchange. In this case, new set of ECDH public keys are not exchanged through Type 6 EAP-NOOB messages. Now, for key derivation, the concatenation key derivation function will take Kz as the input instead of previously derived ECDH shared key. The rest of the inputs for the function are: ASCII string EAP-NOOB as algorithm ID, Np2 as PartyUInfo and Ns2 as PartyVInfo. Similar to the Completion exchange, the out of band nonce (Noob) is not used in this method of key derivation. This method of re-keying is less computationally intensive, but, it does not provide any forward secrecy.

3.5 Error Handling

In this protocol, error conditions are notified by sending a Type 0 error message. Every error message must contain an error code. However, the parameter describing the error (ErrorInfo) is optional. An error message can either be sent from a server or from a peer. After sending or receiving an error message, the server should always terminate the session with an EAP-FAILURE message. Error message transfers can be seen in Figure 9 and Figure 10.

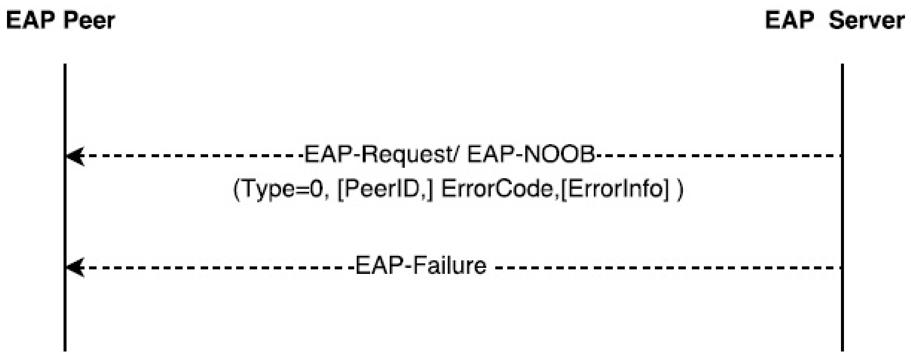


Figure 9: Error notification from server

3.5.1 Error Scenarios

Here, we provide descriptions for the scenarios where either a peer or a server can end up in an erroneous state.

- Invalid message: If the message structure, message length or content of the received EAP-NOOB message is invalid, then an EAP-NOOB message with a suitable error code is sent out. The error code 1001 is used when the NAI or the structure of the NAI is incorrect. The protocol uses JSON scheme of message encoding and decoding. If the received message with a JSON object cannot be parsed or if the object does not contain certain parameter, then error code 1002 is sent. If any of the JSON object member has an invalid

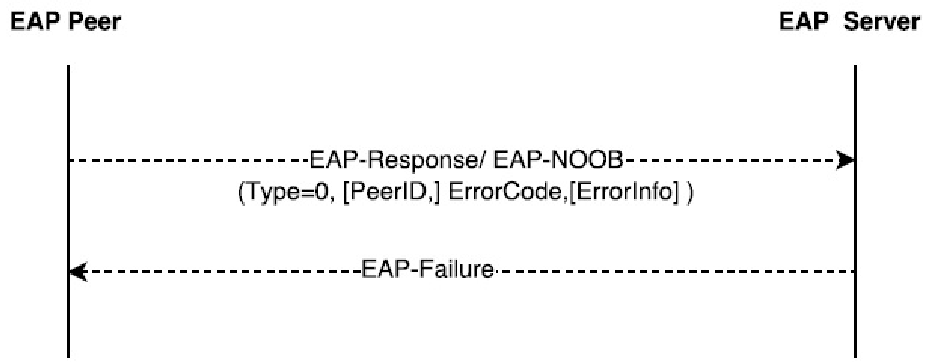


Figure 10: Error notification from peer

length, then error code 1003 is sent out. The error code 1004 is sent if the message type for the received message is invalid. If the peerID in the message is invalid, then error code 1005 is sent. Similarly, If the ECDH Public key component is invalid, then error code 1006 is sent.

– Unwanted peer: To avoid persistent polling for an OOB message, the server assigns a wait time for the peer. The assignment of wait time is limited to implementation specific number of times. Then, for subsequent polling, the server replies with a 2001 error code. After receiving the error code, the peer should stop probing the server. However, the peer is still allowed to try associating with the same server but not immediately. Both server and peer are recommended to notify the error to the user.

– State mismatch : For all the EAP-NOOB messages sent outside the defined set of state machine transitions, error code 2002 is sent. For example, receiving a Type 5 EAP-NOOB request in the *Unregistered state*. As part of recovering from this error, device reset may be necessary. Hence, upon receiving this error, user should be notified about the error.

– Negotiation Failure: The error scenarios relating to the parameter negotiation in both Initial exchange and Reconnect exchange are handled here. In both the message exchange sequences, the peer has to match and select values from the received set of parameters from the server. Therefore, negotiation failure errors are only sent from the peer. The error code 3001 is sent when there is no common protocol version. The error code 3002 is for no common crptosuite and error code 3003 is sent when there is no common OOB direction. A peer may not be able to recover from negotiation failure without a hardware or software upgrade. Hence, it is recommended to notify the user when there is a negotiation failure.

– Cryptographic verification failure: In case of a verification failure of a message authentication code (Hoob, MACp, MACs, MACp2 and MACs2), error code 4001 is sent as a reply. Same error code can be used, if the content of the received OOB message is incorrect. In server-to-peer direction OOB message transfer, if the peerID in the OOB message is incorrect, then it is likely that, a user has unintentionally selected the wrong device for authentication. In that case, it is recommend to notify the error to the user. This is to indicate a user to retry for authentication. During the Completion exchange, when there is a verification failure, both the peer and server should remain in their previous state. Similarly, when there is a verification failure during a Reconnect exchange, both the entities should move to *Reconnecting* state. To retry for the authentication with a new OOB message, a user may have to reset the device.

3.6 Security in EAP-NOOB

The EAP-NOOB protocol provides two fold authentication features. First, the OOB message nonce (Noob) used for generating a shared key will also mutually authenticate the involved parties. Apart from generating the shared key, the nonce is also used for generating the message authentication codes (MAC) on each side. Mutual verification of MAC will confirm the presence of same shared key. However, the Noob alone can not confirm the participating in-band entities to the out-of-band entities. Eavesdropping on the OOB channel can compromise the Noob and can make Man-in-the-middle(MITM) attack possible on in-band channel.

The cryptographic fingerprint in the OOB message (Hoob) for integrity check is the second authentication feature in EAP-NOOB. Altering the contents of a user assisted OOB channel is more difficult than just eavesdropping for the content of the OOB channel. The inclusion of Hoob in the OOB message can confirm the integrity of ECDH key exchange at the receiving side. This will also avert the MITM attack on the in-band channel. If at all an adversary was successful in launching a MITM attack on the in-band channel, and was also able to alter the OOB message, the receiver will still reject the message because of the mismatching Hoob. The confirmation about the integrity of the OOB message is only in one direction. Therefore, in case of a failure, the OOB message sender will be still believe that it has been associated. However, after delivering the OOB message, a user can check for failure by monitoring the protocol stages of both the entities. In case of a failure, the user will notice the single sided association. Then the user can reset the device to reattempt for association. Any explicit mechanism to confirm the integrity of an OOB message to the OOB sender is not included in EAP-NOOB.

The protocol relies on uniquely identifying a device. The contents of parameters, peerInfo and serverInfo will aid a user to identify a server or a peer. As both the parameters are exchanged in plain text, it is recommended not to include any important information such as credentials or device properties as part of the parameter. Rather, include the contents which will help to identify the device. Typical examples include Manufacturer name, Serial number or a web server URL. A fake peer may include false information as part of its peerInfo. Therefore, before moving the peerinfo into persistent storage, it is recommended to have user approval over the collected peerInfo.

Identifier squatting attacks are ineffective in EAP-NOOB. The pseudo-random peerId assigned by the server to a peer, is only valid and confined to the associated server. Association with different server will yield different peerIds. In the case of an OOB message accidentally getting delivered to multiple peers, only the peer which has built up a context with the server by doing Initial exchange will complete the authentication procedure, rest of the peers will be unaffected.

In EAP-NOOB protocol, forward secrecy is optional. During the Reconnect exchange, if only a new cryptographic algorithm is negotiated or new ECDH public keys for an existing cryptographic algorithm is exchanged, then new set of master keys are derived. Deriving a new key ensures the perfect forward secrecy. However, while negotiating a new cryptographic algorithm, it is recommended that both server and peer should look for downgrading attacks related to weaker cryptographic algorithms, and should not renegotiate into a deprecated or vulnerable cryptographic algorithms.

4 Implementation

In this chapter, we will describe the EAP-NOOB protocol implementation process. The chapter starts with an overview of the protocol implementation. Next, we explain the procedure to register a new EAP method. This is followed by an explanation of the message sequence implementation. Later in the chapter, we describe the web server implementation and automation of the authentication process. This is followed by a description of the code changes done for controlling the source package. Finally, we will end this chapter by mentioning a design change made in the protocol to solve a user identification problem.

4.1 Overview

The EAP-NOOB protocol is developed and integrated into well tested open-source projects. There exist two well known open source supplicant implementations, WPA_Supplicant from the HostAP project and Xsupplicant from the Open1X project. Both the projects have focused on UNIX like operating systems and are licensed under the BSD and GNU general public license. Also, most of the existing 802.1X/EAP authentication methods are supported by both projects. Upon considering the frequency of updates, number of users and size of the support community, we chose WPA_Supplicant as the source project for our EAP-NOOB peer implementation. Along with the WPA_Supplicant, we also selected Hostapd. The Hostapd is an open source implementation of the back-end authentication server. It was chosen for our EAP-NOOB server side implementation. It is also managed by HostAP project, and hence the same factors which influenced in selecting WPA_Supplicant were considered when selecting Hostapd.

The authenticator in EAP-NOOB is only used in pass-through mode and also no part of the EAP-NOOB implementation includes modification of the authenticator. Thus, any EAP-Enterprise supporting authenticator can be used in the EAP-NOOB development environment and in eventual deployment. The components involved in implementing the EAP-NOOB protocol can be listed as:

- WPA_Supplicant: Support for the EAP-NOOB method on the peer side will be implemented as extensions to this software package. The existing support for the wireless network association and EAPOL requires no change.
- Hostapd: The EAP-NOOB server will be implemented as extensions to this software package. The core part of the RADIUS authentication server and EAP server already exists in the Hostapd package.
- Web server: The web server works in tandem with the EAP-NOOB server and also shares a database with it to maintain the EAP-NOOB peer contexts. The user can track the authentication process through the web server. Also, it acts as one end of the OOB channel; the OOB message directed towards the EAP-NOOB server will reach the web server first, and the OOB messages

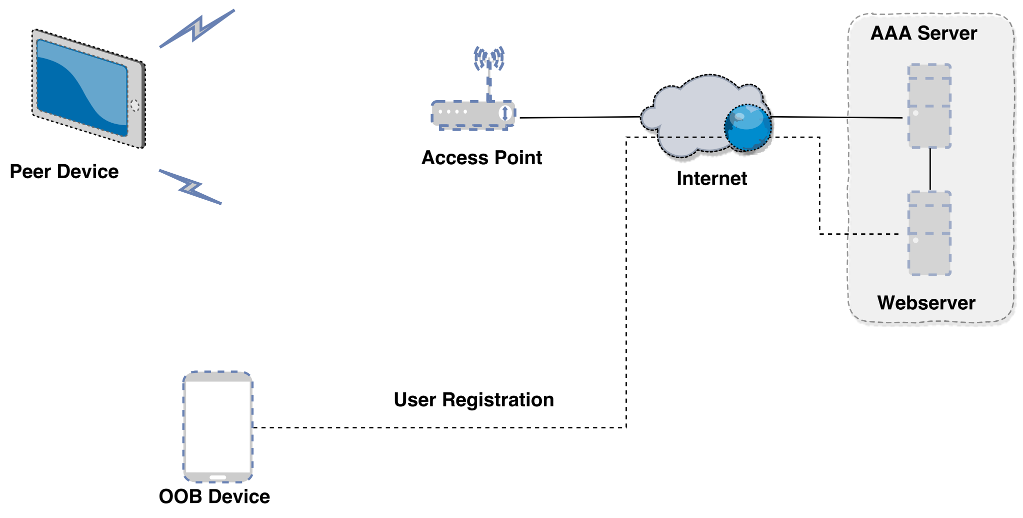


Figure 11: EAP-NOOB Setup

originating from the EAP-NOOB server will be delivered through the web server.

- OOB device: A device which assists in passing a message over the OOB channel. The selected device should contain the tools and applications necessary to send and receive the OOB message. In this implementation, a smart-phone was used as an OOB device. A smart-phone is commonly available and it can automate the message passing process through its inherent features such as scanning barcodes through a camera, displaying a message or connecting to a web server over the Internet.

The OOB device is considered an optional component for the protocol specification. However, considering the familiarity of Smartphone to any average user, we included the OOB device as one of the components int the EAP-NOOB implementation. Then, we defined the OOB message format as a Universal Resource Locator (URL) with *base64url* encoded Hoob, Noob and peerID parameters.

The EAP-NOOB setup in figure 11 can be further extended by introducing a local AAA server. Typically access network has a local AAA server. This new component will be placed between the authenticator and the back-end AAA server. It will either handle the authentication procedure on its own or relay the messages between the authenticator and the back-end Hostapd server. Introducing an intermediate AAA server extends the hierarchy in the authentication process and is useful in scenarios involving supplicant roaming. Considering the current version EAP-NOOB specification [42], extending the authentication hierarchy may not be necessary in this implementation. Hence a local AAA server in not included as a component. Alternatively, when the back-end AAA server is operating in the cloud, authentication request from the access network will arrive first at the local AAA

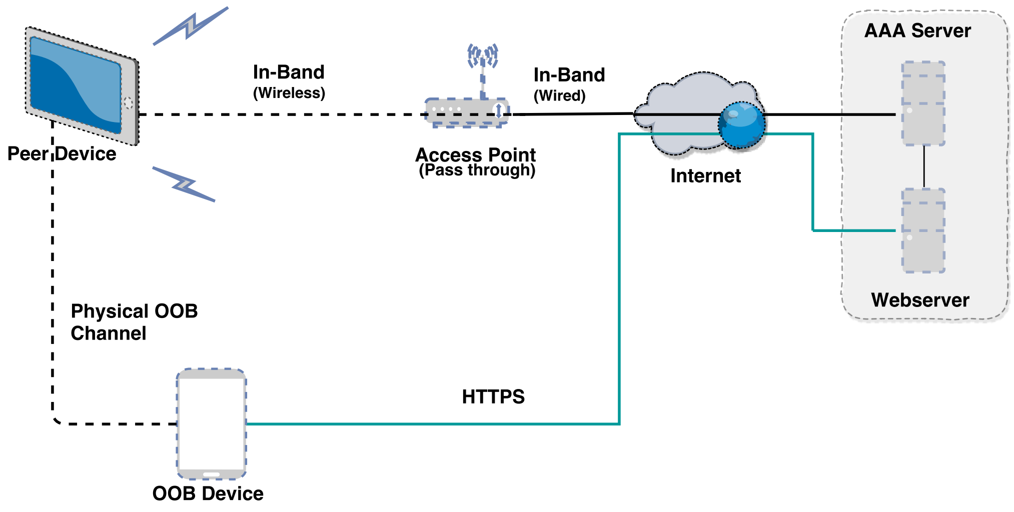


Figure 12: In-band and Out-of-band channels

server. The forwarding rules in the local AAA must be set to relay the messages to the back-end authentication server.

4.1.1 OOB Message Transfer

In the current protocol implementation, we use URL as the OOB message format. It is transferred either by encoding it as a QR-code or as a NDEF record through Near Field Communication (NFC). Our implementation supports three different methods of passing an OOB message and, they can be listed as:

- peer-to-server with QR-code.
- peer-to-server with NFC.
- server-to-peer with QR-code.

4.1.2 About Source Packages

The WPA_Supplicant is designed to be a daemon program, which is used to configure the wireless network interface at client station side. The software package contains the WPA supplicant component and can perform WPA key negotiation and EAP based authentication. It works across different platforms, including UNIX-like and Windows operating systems. The software supports a variety of WLAN drivers (MadWifi, Hostap, Atmel, Broadcom, .etc) and has both command line and graphical user interface (GUI) for managing the supplicant. WPA_Supplicant supports both EAP-PSK and EAP-Enterprise based authentication. However, in our implementation only EAP-Enterprise authentication is used.

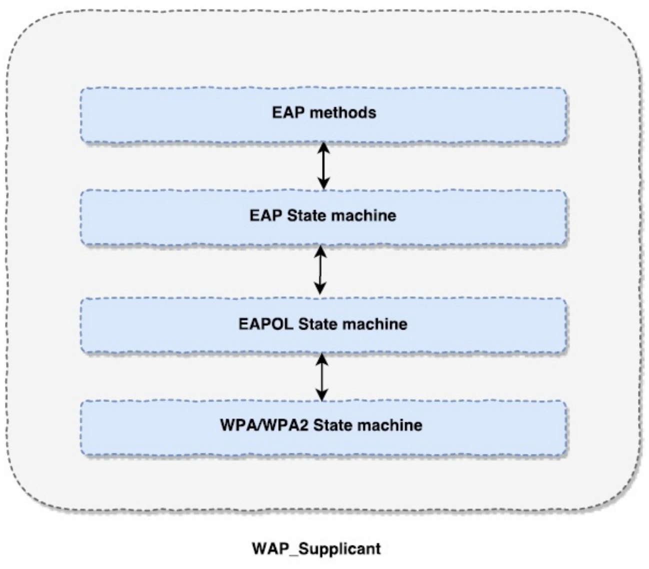


Figure 13: EAP architecture

Now, looking at the architecture of WPA_Supplicant, each EAP method is considered as a separate module and the methods are implemented on top of the EAP state machine. The EAP state machine resides on top of the EAPOL layer, which maintains its own state machine to track message transfer over the EAPOL. Each EAP method packet will be encapsulated in an EAP packet which in turn will get encapsulated in an EAPOL packet. Figure 13 only covers a part out of WPA_Supplicant architecture. As the supplicant is a full-fledged wireless network configuration system supporting both open and secure wireless connection establishment, we have only considered the part of the architecture which involves 802.1X components.

Similarly, the Hostapd is also designed to be a daemon programme and usually acts as a back end component for controlling the authentication procedure. It contains the implementation for 802.1X/EAP authenticators, RADIUS client, EAP server and RADIUS authentication server. Similar to WPA_Supplicant, Hostapd also supports both EAP-PSK and EAP-Enterprise authentication, and it contains implementations for the WPA/WPA2-PSK authenticators. In this implementation, we use Hostapd only for the 802.1 X/EAP authentication. The Hostapd can be configured either through GUI or using command line interface. As both Hostapd and WPA_Supplicant are from the same project, the EAP parts of the architecture at both the entities are similar. Hence the EAP architecture from Figure 13 is also relevant for Hostapd. For the sake of simplicity, the complete architecture of both the projects is not explained here; instead only the relevant parts are presented.

4.1.3 Development Environment

Linux is the most commonly used operating system for embedded smart devices. Also, both Hostapd and WPA_Supplicant are more focused on supporting the Linux platform. Although WPA_Supplicant does support the Windows platform, only the Linux version of the software package has a large user community. Considering the popularity of Linux among IoT smart devices and also the support from the selected software packages, the complete implementation of our EAP-NOOB protocol is done on the Linux platform.

Operating system	Ubuntu 16.04
Primary memory	8 Gigabytes
HDD	320 Gigabytes
Processor	Intel Core i5 vPro

Table 1: Development machine specification

The source package WPA_Supplicant and Hostapd are implemented in the C programming language; therefore our implementation of EAP-NOOB peer and server were also done in the same programming language. We have developed the web server on Node.js[3]. Node.js was chosen because it is fast and often used for developing real time applications [46]. It also has an easy-to-use package manager (npm[4]). Later, we automated the complete peer authentication process, including two directions of OOB message passing in Python3[5]. The specification of wireless work stations (laptops) used for implementation and testing the EAP-NOOB protocol is given in Table 1.

4.2 Registering the EAP-NOOB method

In both Hostapd and WPA_Supplicant, we have developed the new EAP method as a separate module over EAP. In both the source packages, there exist a common interface between all the EAP methods and the EAP state machine. The interface allows us to implement any new EAP method without modifying the underlying EAP state machine. Initially, the new EAP method is expected to implement the functions defined in the interface *struct eap_method* in *eap_i.h*. The structure only contains pointers to the necessary functions. The new method should implement the relevant functions from the structure and assign the pointers in an explicit registration function. In our implementation of EAP-NOOB, function *eap_peer_noob_register()* is defined to register the method name and type, and also to initialize the EAP interface. Also, the EAP-NOOB method registration function

[3]https://nodejs.org/en/
[4]https://www.npmjs.com/
[5]https://www.python.org/

should be included into the EAP function *eap_register_methods()*, which is responsible for registering the EAP-Method during program execution. Additionally, the method name should also be included in the supplicant Makefile to build the new method while compiling the supplicant executable. The common interface for the EAP methods is defined in section 4.4 of RFC 4137 for implementing EAP state machines [41].

4.3 Message Sequence Implementation

To comply with the protocol specification, we form the EAP-NOOB messages as unicode (UTF-8) encoded JSON objects. To encode and decode EAP-NOOB messages, APIs from Jansson[6] are used. Jansson is a JSON library for C programs and provides full unicode support. It is an open source library licensed under the MIT licence. In both EAP-NOOB peer and server, we have defined a separate functions for encoding and decoding messages. Also, the parameters Vers, Dirs, Cryptosuites and contents of serverInfo from Type 1 EAP-NOOB request are made configurable by introducing a configuration file *eapnoob.conf* at the server side. Similarly, a configuration file with the same name at peer side is introduced to make parameter Verp, Dirp, Cryptosuitep and peerInfo configurable. The respective message encoding functions will read the configuration file before encoding Type 1 Request/Response messages. Rest of the parameters in EAP-NOOB are non-configurable or are calculated internally during the course of the authentication

In WPA_Supplicant, the decapsulated EAP payload from the EAP-state machine is transferred to the respective EAP method through a designated function. The function assigned to the function pointer *process* from interface *struct eap_method* will receive the EAP payload from the EAP state machine. At the peer side, *eap_noob_process()* is our designated function to receive messages, and after processing the received message a suitable response is sent from same function.

Similarly in Hostapd, the de-capsulated EAP payload is transferred to a designated function. However, the message handing at the server side is different from the peer side implementation. The EAP state machine at server side first invokes the EAP-NOOB function *eap_noob_check()*. Here, the validity of the messages with respect to the EAP-NOOB state machine is verified. A successful validation of the message will make EAP state machine to call the function *eap_noob_process()* and the function will decode and handle the received EAP-NOOB message. Also, the next message to be sent will be decided here. Finally, after successful message processing, the EAP state machine will prompt function *eap_noob_buildReq()* to build next message from the EAP method. Although, the EAP-NOOB state machines at the server and peer are the same, the underlying EAP state machine is different at both the sides. At the peer side, if *EAP-peer* state machine is in use and at the server side, EAP makes use of *EAP-backend-authenticator* state machine

[6]`http://www.digip.org/jansson/`

[41]. This difference in state machines makes the server and peer use different execution paths for handling the EAP-NOOB messages. However, the functionality of the before-mentioned three EAP-NOOB server-side functions is covered by a single function *eap_noob_process()* at the EAP-NOOB peer side. This is essential in order to match the EAP-NOOB server and peer state transitions. A representation of message handling at both the entities is presented in Figure 14.

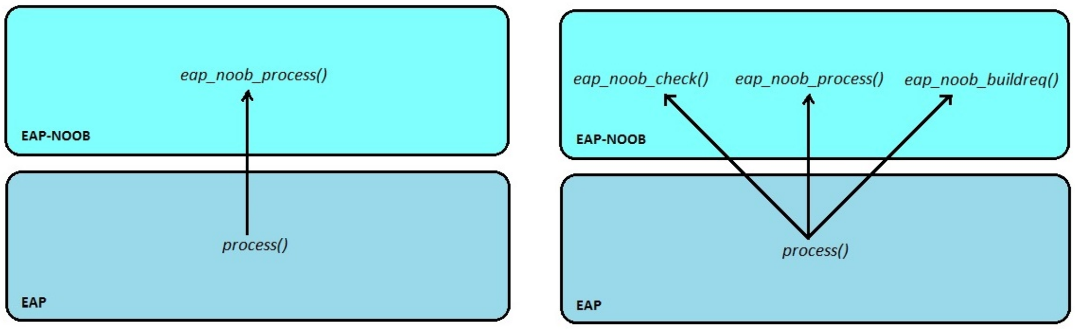


Figure 14: Message handling: supplicant vs server

4.3.1 Cryptographic Library

In our implementation, we have used the application programming interfaces (API) from the OpenSSL[7] library for all the cryptographic operations, including:

- Generation of Private-public key pair
- Derivation of shared key
- Generation of Nonces
- Calculating cryptographic fingerprints
- Calculating message authentication codes

OpenSSL is a general purpose open source library, licensed under its own OpenSSL and SSLeay licenses. It is a popular and well tested cryptographic library. It offers commercial-grade and full featured support for developing software solutions which involve cryptographic operations.

4.3.2 Persistent Storage

At both sides, the received and the calculated values as parameters are stored to a database. As EAP-NOOB spans multiple EAP session, at the end of each EAP session, the data stored in the EAP-NOOB memory will be freed. Hence, moving

[7]https://www.openssl.org/

data to a database before the end of every EAP session is essential. In our implementation, the Sqlite3[8] database is used for storing data. Sqlite3 is an in-process library containing a transactional database engine. The library provides interfaces for C/C++ programs and is renowned to be fast and reliable.

4.4 Web Server

A device performing 802.1X authentication is only expected to exchange EAP messages with the authentication server. However, in the EAP-NOOB protocol, an out-of-band message should me transferred to complete the authentication. Therefore, in our protocol there is the need for an entity interfacing with real world to send or receive the out-of-band message. As part of our server-side protocol implementation, we developed a web server module. The primary objective of the web server is to act as a mediator between the EAP-NOOB server and the OOB channel in the real world. In the peer-to-server direction of the out-of-band message passing, the user will deliver the OOB message to the web server, which transfers the contents of the message to the EAP-NOOB server. Similarly, in server-to-peer direction of message passing, the web server will read the OOB message from the EAP-NOOB server and will present it to the user for delivering it to the peer device.

Apart from authentication, the protocol EAP-NOOB also involves maintaining and managing the associated devices. In order to do that, it is essential to establish the ownership of devices. Apart from being a mediator for the server, the web server also help to establish the ownership of the authenticating device. Without the web server, any Wi-Fi enabled device can get associated with the EAP-NOOB server. So, always an EAP-server will work in tandem with a web server. Also, any user attempting to associate a device with a EAP-NOOB server will first have to have an account with the web server. After the authentication, the devices are always registered under a user account. Irrespective of the selected OOB direction, the ownership of the authenticating device is always established during the OOB message transfer. The ownership establishment procedure in each direction can be listed as:

– peer-to-server: Initially, the OOB message is received at the user device. The URL from the OOB message will lead the user to the web server. Here, a user should have a valid user account, without an account, the web server will reject the OOB message. After delivering the message to a valid user account, the authenticating device will be associated to the account through which the OOB message is delivered.

– server-to-peer: First, the correct device which has completed the Initial exchange should be identified from a user account. The identification is based on the unique device identifier shared through the peerInfo parameter. The user will search for the device and will add it to the user account. Adding the device and completing the OOB step will confirm the ownership of the device.

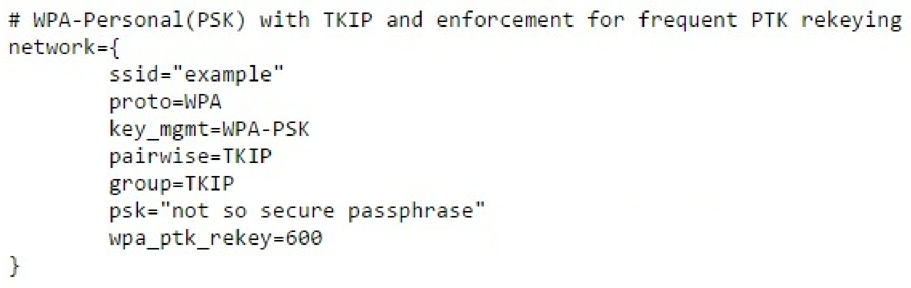
[8]https://www.sqlite.org/

The web server introduces a level of transparency into the authentication process. During authentication, a user can see every state transition at the EAP-NOOB server from the web server. Also, errors during authentication are notified through the web server.

The web server is developed in Node.js with embedded Javascript views. The EAP-NOOB server and the web server use a common database to maintain user and device related information. However, other than accessing and reading a common database, there exist no other means of communication between the server and the web server. A web server should always have a trusted third party certificate to prove its authenticity to the user and to the OOB devices. The serverInfo parameter passed in the Initial exchange contains the URL of the web server and the same will be used while forming the OOB message. If it is identified that the URL does not lead to a web server with a secure HTTP connection, the peer will consider both the EAP-NOOB server and the web server as untrusted and will terminate the authentication procedure by sending an error message (Error number 1003). If a self-signed certificate or certificates issued by an untrusted root are used by the web server, the user will be notified by the web-browser while transferring the OOB message.

4.5 Automating the Authentication

Conventionally, the WPA_Supplicant attempts to associate with APs based on the network block information. Each AP is represented as a network block and the configuration information necessary to associate with the AP is grouped in a network block. The supplicant can receive the network block information either through the command line using wpa_cli or by reading a designated configuration file. If more than one network block is received at the supplicant, the association with each of the network block is attempted sequentially until the supplicant gets associated with one of the APs. An example network block can be seen in Figure 15.



```
# WPA-Personal(PSK) with TKIP and enforcement for frequent PTK rekeying
network={
        ssid="example"
        proto=WPA
        key_mgmt=WPA-PSK
        pairwise=TKIP
        group=TKIP
        psk="not so secure passphrase"
        wpa_ptk_rekey=600
}
```

Figure 15: Example network block

The EAP-NOOB protocol is applicable to devices with variable user interface capabilities. Now, either due to limited user interface capabilities of the device or due to the nature of the selected source packages, configuring devices using the current implementation originally required a few manual operations. Instead, automating the complete authentication procedure will enhance the usability of the protocol implementation.

As part of our implementation, we have automated the authentication procedure from the EAP-NOOB peer side. After the automation, the user will just deliver the OOB message and need not do any other manual operation. The automation includes support for authentication in both directions. As part of automation, a peer device will be first initialized. The lists of activities done as part of the initialization is as follows:

- Scanning to identify the nearby access points (AP)
- Identifying WPA2-Enterprise supporting APs and sorting them by the received signal strength
- Creating a new configuration file with a network block for each of the member from the list
- Starting WPA_Supplicant with the configuration files.

The supplicant will perform the Initial exchange with all the APs which are connected to an EAP-NOOB supporting backend authentication server. Based on the selected OOB direction, the peer will be prepared to either deliver or receive the OOB message. When the selected OOB direction is peer-to-server, an OOB message can be delivered either as a QR-code or as a URL through NFC. Now, if the selected OOB message is a QR-code, then the peer displays a QR-code for every EAP-NOOB server with which the peer has performed an Initial exchange.

As the WPA_Supplicant attempts for AP associations sequentially, it is hard for our implementation to obtain a complete array of QR-codes instantaneously. Therefore, the user will have to wait a short amount of time to see all the QR-codes. When all the QR-codes are displayed, the user will then scan the appropriate one to authenticate the device with the configuring network and server. Transferring a valid OOB message to the correct server will initiate the Completion exchange. A QR-code as the OOB message can be used with the device which only has output interface capability. A display unit can be considered as an example device.

Similarly, when the selected OOB message is a URL through NFC, our implementation only supports transferring OOB messages for a single server. Therefore, there should only be a single AP in the vicinity which is connected to an EAP-NOOB supporting backend server. The NFC channel is included as an alternative to scanning or reading OOB message from an output device. However, this alternative method of message transfer still need to be enhanced to make it as fully fledged as QR-Code based OOB message transfer.

When the selected OOB message direction is server-to-peer, the peer is expected to receive the OOB message. In our implementation, at the peer side, a QR-code code scanner will be activated. In this implementation, an open source application named Zbarcam [9] is used to read QR-codes at the peer side. The application will start the camera unit of the device and will be waiting to read any QR-code message. When the user holds the OOB device with QR-code sufficiently close to the camera unit, the Zbarcam application will read the OOB message from the OOB device. Then, a valid OOB message will initiate the Completion exchange.

In both the OOB message directions, if the authentication procedure ends with EAP-SUCCESS, the peer is prompted to check for a new IP address for its wireless network interface. Obtaining the IP address from an AP confirms that the peer device is part of a WLAN. In our implementation we use Linux system call *dhclient*[10] to obtain the IP address. Finally, to test the Internet connectivity, we play a video from a popular video-sharing website, Youtube[11]. The successful reception of the video content confirms the peer device's network connectivity. In WAP_Supplicant, the complete source code for automating the authentication procedure is placed in a Python3 script named wpa_auto_run.py.

4.6 Controlling WPA_Supplicant

WPA_Supplicant is a well structured software package written by a long list of contributors[12]. The software architecture is modular and the inter-module communication is well defined. The existing EAP methods were designed to have no control over the supplicant behavior. However, to implement certain functionality required by our EAP-NOOB protocol, the supplicant is expected to receive commands from the new EAP method.

Now, considering the segment of the supplicant relevant for introducing a new EAP method, the interface between the EAP module and the new EAP method is strictly confined to *struct eap_method*. Additionally, the new method is only allowed to access common utility functions. This arrangement is for reducing the burden of any new method and also to encourage code modularity and re-usability. However, while implementing the EAP-NOOB peer, we discovered that certain features of the protocol cannot be implemented without making previously defined inter-module references. This mainly involves, accessing or modifying data structures and calling functions from modules with which there previously existed no direct connection. In the next section, we will describe all the inter-module references made in the protocol implementation.

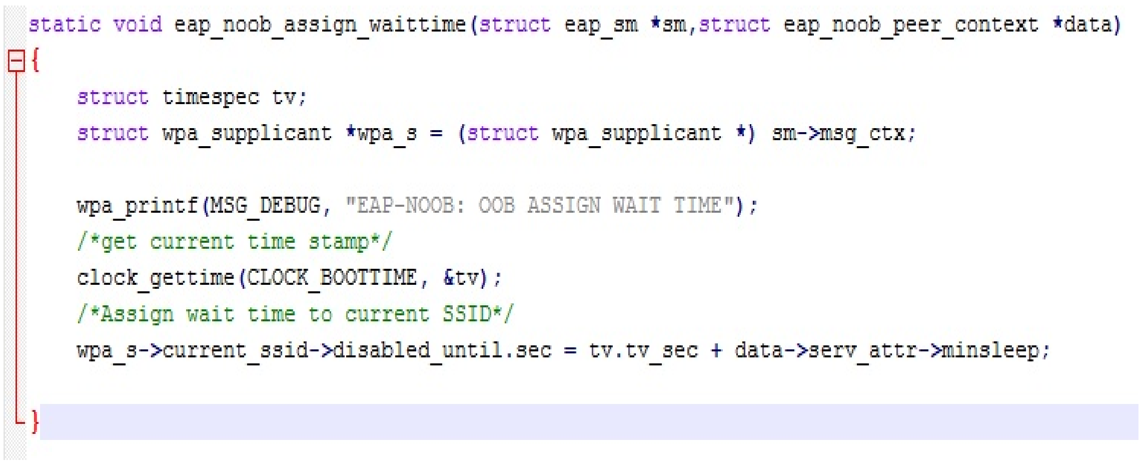
[9]http://manpages.ubuntu.com/manpages/wily/man1/zbarcam.1.html
[10]https://linux.die.net/man/8/dhclient
[11]https://www.youtube.com/
[12]https://launchpad.net/wpasupplicant/+topcontributors

4.6.1 Wait Time Assignment

When the EAP-NOOB server assigns a wait time to a peer in the Waiting exchange, the peer should probe only after the assigned wait time has elapsed. For example, if the server assigns 60 seconds as the wait time, and then the peer should at least wait for 60 seconds before starting the next Waiting exchange. However, this feature could not be implemented just with the existing interface to the EAP state machine. There exists no function in the *struct eap_method* with which a peer can be stalled for a time equivalent to wait time. Architecturally, stalling the EAP method by implementing a waiting loop is not the right solution. A peer may have to do the Waiting exchange with multiple servers at the same time, and stalling a peer will prevent that. Therefore, an alternate way for implementing the wait time was needed.

WPA_Supplicant On the other hand, has a provision to temporarily disable association attempts to a service set identifier (SSID). The supplicant data structure *struct wpa_ssid* holds the network configuration data for each SSID. It has a member named *struct os_reltime disabled_until*. The member variable can be assigned a time stamp. The SSID will be considered unavailable until that time. Each of the network blocks from the designated configuration file is mapped to an instance of *struct wpa_ssid*. This arrangement can be used to temporarily disable associations with any of the network, if at all it is necessary. The contexts maintained for each SSID is in a *struct WPA_Supplicant* data structure. The data structure also holds the configuration and run time variable data for the WPA_Supplicant interface. The WPA_supplicant context is not directly related to a new EAP method. In fact the supplicant context holds a context for EAPOL, and the EAPOL context maintains the EAP state machine. Finally, the EAP state machine will have the context for the EAP method. Because of the many layers of abstraction, the EAP method is not expected to control the WPA_Supplicant data. Nevertheless, to have a working prototype, we access the relevant member of *struct wpa_ssid* to assign the waiting time.

The data structure for the EAP state machine (*struct eap_sm*) is already accessible from any EAP method. The data structure contains pointers to both the EAPOL and WPA_Supplicant context. To assign the wait time, the pointer to supplicant data is accessed. Now, from the supplicant data structure, one can access the data structure of the current SSID. Later, we calculate the time stamp as the sum of the current time and the wait time received from the server in the Type 3 EAP-NOOB request. We assign the time stamp to the member *disabled_until* from current SSID context. After the assignment, the supplicant does not try for an association utill the end of the wait time.



```
static void eap_noob_assign_waittime(struct eap_sm *sm,struct eap_noob_peer_context *data)
{
    struct timespec tv;
    struct wpa_supplicant *wpa_s = (struct wpa_supplicant *) sm->msg_ctx;

    wpa_printf(MSG_DEBUG, "EAP-NOOB: OOB ASSIGN WAIT TIME");
    /*get current time stamp*/
    clock_gettime(CLOCK_BOOTTIME, &tv);
    /*Assign wait time to current SSID*/
    wpa_s->current_ssid->disabled_until.sec = tv.tv_sec + data->serv_attr->minsleep;

}
```

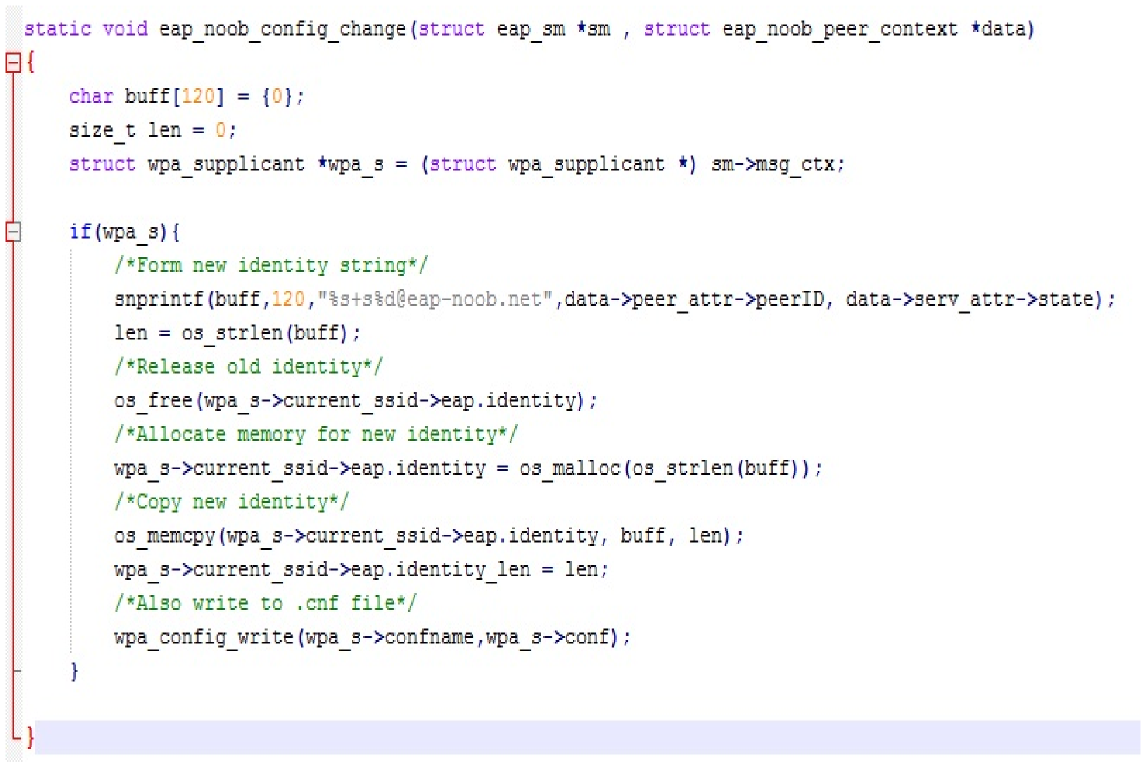
Figure 16: Wait time assignment code snippet

4.6.2 Updating Peer State

In EAP authentication, when a supplicant is initially queried for its identity through EAP-Identity/Request message, the supplicant responds with an EAP-Identity/Response containing its own identity. Each network block in the supplicant configuration file may specify an identity for the network. If there is no identity specified, anonymous identity is assumed for the network. In case of EAP-NOOB, initially, every network block will have *noob@eap-noob.net* as the supplicant identity, this is the generic NAI used by the protocol. However, when the peer EAP-NOOB state machine transits to a different state, the identity for the respective network should also be changed. The current NAI in EAP-Response/Identity helps the EAP-NOOB server to be aware of the EAP-NOOB peer state. Knowledge of the peer state is essential for deciding the next state transitions.

Now, at peer side, the identity for each network is stored in an instance of the data structure *struct wpa_ ssid*. The instance is allocated for the each network block in the configuration file. All the allocated instances reside in the main supplicant data structure named *struct WPA_ Supplicant*. As there is no predefined interface between an EAP method and the WPA_Supplicant module, the identity of a network block cannot be directly accessed from the EAP-NOOB context. Therefore we made an indirect inter-module reference to update the identity of a network

As was the case with wait time assignment, the current SSID context present in the data structure *struct WPA_ Supplicant* is accessed through the EAP state machine data. The member *struct eap_ peer_ config* present in the current SSID context holds the supplicant Identity for the network through its member variable *u8 * identity*. After every state transition in the EAP-NOOB peer, the identity is changed to the format *peerID+sX@eap-noob.net*, where *s* is the current peer state and *X* is a number between 0 and 4. Changes made to the variable *u8 * identity* will



```
static void eap_noob_config_change(struct eap_sm *sm , struct eap_noob_peer_context *data)
{
    char buff[120] = {0};
    size_t len = 0;
    struct wpa_supplicant *wpa_s = (struct wpa_supplicant *) sm->msg_ctx;

    if(wpa_s){
        /*Form new identity string*/
        snprintf(buff,120,"%s+s%d@eap-noob.net",data->peer_attr->peerID, data->serv_attr->state);
        len = os_strlen(buff);
        /*Release old identity*/
        os_free(wpa_s->current_ssid->eap.identity);
        /*Allocate memory for new identity*/
        wpa_s->current_ssid->eap.identity = os_malloc(os_strlen(buff));
        /*Copy new identity*/
        os_memcpy(wpa_s->current_ssid->eap.identity, buff, len);
        wpa_s->current_ssid->eap.identity_len = len;
        /*Also write to .cnf file*/
        wpa_config_write(wpa_s->confname,wpa_s->conf);
    }

}
```

Figure 17: Configuration change code snippet

only modify the supplicant identity in the process memory. However, updating the identity parameter of a network block, present in the supplicant configuration file would enable the peer to continue the authentication process inspite of restarting of the the WPA_Supplicant. To achieve this we used the function *wpa_ config_ write()* defined by the WPA_Supplicant module. The function is meant for rewriting the network configuration into a configuration file. The code snippet used to change the identity at both the process memory and the configuration file can be seen in Figure 17.

4.7 Hint Message for User Identification

To authenticate a device with server-to-peer as the OOB message direction, the user will first have to identify the device at the web server portal. The device identification can only be based on the information shared by the peer as part of peerInfo parameter. Until the user delivers the OOB message through a valid user account, the identification and selection of a device at web server is not bound to any user. In a scenario where multiple users are authenticating a device at the same time, more than one OOB message may be sent to the peer, and the peer will only accept the first one. The server will be aware that more than one OOB was sent to the peer. However, the server cannot determine which OOB message in use at the peer side.

In the current version of the protocol, the Completion exchange is initiated soon after an OOB message is received at either of the entities. During this phase, the message authentication codes relating to only one OOB message can be verified. Failure of the verification will not allow trying verification for other sent OOB messages. Therefore, we modified the Completion exchange for the server-to-peer direction to resolve the issue when there are multiple users trying to associate with the same device.

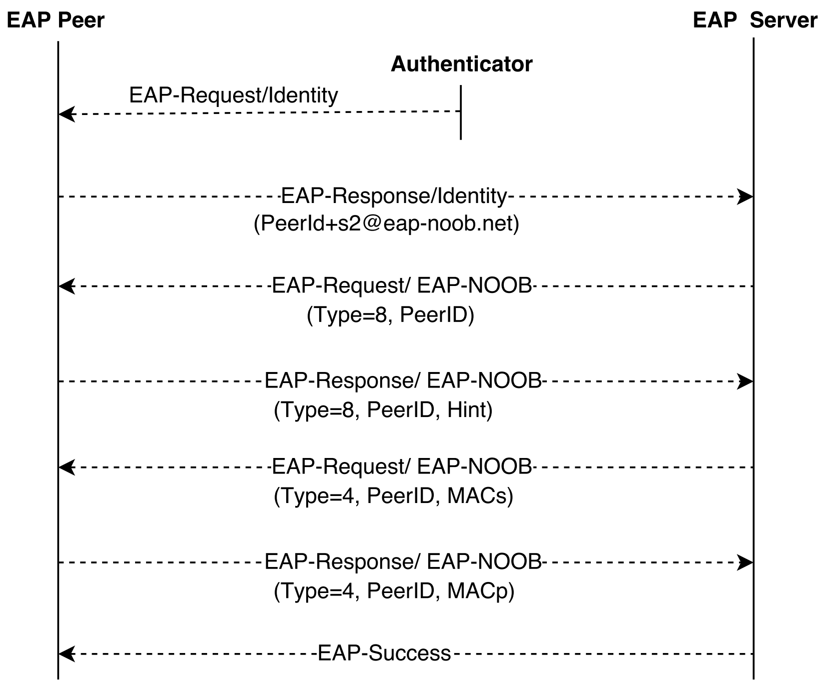


Figure 18: Modified Completion exchange

To rule out the ambiguity of choosing the owner of a device, we have introduced a new hint message in our implementation. At the beginning of the Completion exchange, the server will request for a hint from the peer to identify the OOB message parameters in use. The hint is requested through a separate Type 8 EAP-NOOB message. The peer will generate a cryptographic hash using the *Noob* parameter and a static salt value. The hint is sent over the Type 8 EAP-NOOB response.

When the selected direction is server-to-peer, for every OOB message generated for a device, the web server will also generate the hash of *Noob* along with the same salt value as peer side. Therefore, at the beginning of the Completion exchange, the server will already have a list of hash values based on all the OOB messages. When the hint message is received from the peer, the server will compare the received hint with its own list of cryptographic hashes. Then, the peer will be associated with the context which holds same cryptographic hash as the received hint. Finally, after identification of the owner, the Completion exchange will complete with the mutual authentication procedure

The hint message is not necessary in the peer-to-server direction as the user will have to be in the vicinity of the device to obtain the OOB message from the peer. Presence of more than one user for authenticating the same device can be easily detected in this case. However, in the case of server-to-peer direction, any user can select and try to add the device from the web server portal. This creates the possibility to send more than one competing OOB message to a peer. The modified Completion exchange message sequence can be seen in Figure 18.

5 Discussion

In this section, first we will discuss the security offerings from the protocol. Later, we will discuss the authentication latency for the protocol. Finally, the section will end by presenting a use case scenario for the protocol.

5.1 Results and Analysis

The major factor influencing the usability of the EAP-NOOB protocol is the amount of time necessary to complete the authentication procedure. The protocol involves user assistance at a precise step in the state machine. It is expected that the device will complete the Initial exchange procedure and will be ready to either send or receive the OOB message in a short time. This will make a user to believe that the device is ready for registration with the server right after the device is powered on. However, the OOB message delivery is expected to take variable amount of time, as it is dependent on the complexity of using the OOB channel. With a simpler OOB channel and familiar methods of OOB message transfer, the device authentication can be completed in just few seconds. In our protocol implementation, we have used scanning QR-code or NFC transfer as the OOB channels. Message transfer in either of the channels can easily be done using a smart phone. Now, considering the familiarity of smart phones to most modern day IoT device users, authenticating a device using our implementation of EAP-NOOB protocol should not be difficult.

The total time for authenticating a device using our implementation can be coarsely estimated. It is equal to the cumulative sum of time necessary for in-band message exchanges with the sum of wait time assigned during each Waiting exchange. As the user assistance time overlaps with the wait time, it is not accounted individually. It is seen in the state machine, a device may transit to *Registered* state either from *Waiting for OOB* state or *OOB received* state. Hence, one cannot expect the in band message exchanges to be completed at a constant time. Similarly, user assistance for delivering the OOB message cannot be expected to be completed in a fixed time duration. Therefore, one can only estimate the range of time duration necessary for completing the authentication with the EAP-NOOB protocol.

In this part of the thesis, we will analyze the authentication time for the current protocol implementation. The analysis is based on the authentication time determined by collecting the timestamps at different protocol stages. The time stamps are only collected at the WPA_Supplicant. The protocol testing environment used for collecting timestamps included:

– An authentication server (Hostapd).

– A web server.

– A personal computer with WPA_Supplicant.

– A smart phone.

- Five access points.
- A NFC reader device.

The authentication time is collected for all the supported OOB direction and for all the implemented OOB message types. The total time for authenticating a device is determined based on,

Total time = Method initialization time + in-band message exchange time + out of band message transfer time

Here, the Method initialization time refers to the time between the start of the WPA_Supplicant and until EAP-NOOB method is called for the first time. When there are more than one SSID to be attempted for authentication, the in-band message exchange time is calculated by adding two separate time durations. First, the duration from the first Type one EAP-NOOB message till the generation of the last OOB message is collected. Then, the time duration from the reception of Type four EAP-NOOB messages until the MSK is exported from the method is calculated and added to the first value. Finally, the time duration between the generations of last OOB message until the reception of Type four EAP-NOOB request messages is accounted as the out-of-band message transfer time.

Now consider a scenario where user is authenticating IoT device using our EAP-NOOB implementation. The automation script at the peer side will try association with every EAP-Enterprise supporting APs. When there are more than one EAP-Enterprise supporting AP in the vicinity, a user may not be able to select the relevant SSID until it is prompted either at web server or as at peer side. Then, the authentication latency will depend on number of AP in the vicinity. To simulate such an environment in our test setup, we used up to five SSIDs while attempting for device authentication. The SSID *NOMAD, EAP_NOOB_1* and *EAP_NOOB_2* are connected to same authentication server, which support EAP-NOOB. *eduroam* and *aalto* are the other two SSIDs which support EAP-Enterprise kind of authentication but not EAP-NOOB.

5.1.1 Peer-to-Server

The current implementation supports two kinds of OOB message transfer in the peer-to-server direction: URL as a QR-code and URL through NFC. For URL through NFC, the protocol implementation can only carry out authentication when there is only one EAP-NOOB supporting AP in the vicinity. While testing this mode of message transfer, *EAP_NOOB_1, eduroam* and *aalto* were kept active. Out of three active SSIDs, the device can get authenticated with only *EAP_NOOB_1.* The collected reading can be seen in Table 2.

As there are three SSIDs for the device, the automation script will create three separate network blocks in the supplicant configuration file. Then, upon reading the

SSID list	initialize time	in-band	out-of-band	total time
EAP_NOOB_1 eduroam aalto	32.584	0.788	27.697	61.069

Table 2: Authentication latency for URL through NFC

configuration file, the WPA_Supplicant attempts for association sequentially. The sequence of appearance of network block in the configuration file is based on the received signal strength. Instances where the signal strength of SSID eduroam and aalto are greater than the signal strength of EAP_NOOB_1, the supplicant will attempt the association with the SSIDs in the order eduroam, aalto and finally with EAP_NOOB_1. As both eduroam and aalto does not support EAP-NOOB, the association attempt with both the SSIDs will end with a failure. This will actually delay the authentication procedure. It can be seen from table 2 that the method initialization time is recorded as 32 seconds. It is for the same reason that method EAP-NOOB is only called after the association attempt with the first two SSID ends with a failure.

Initially, while listing the SSIDs, the supplicant will have to consider all the SSIDs which support the EAP-Enterprise mode of authentication. This is because, based on the result from WPA_Scan, the supplicant can only know about the supported mode of authentication by each of the SSID. Therefore, using the available information, supplicant will have to communicate with each of the authentication server connected to the APs in the vicinity to inquire for the support for the EAP-NOOB protocol.

For URL through QR-code method of OOB message transfer, the current implementation supports displaying of QR-code for multiple SSIDs. While collecting authentication time for this method of message transfer, up to five SSIDs were used. Table 3 and Table 4 shows authentication time for scenarios with variable number of SSIDs. Authentication times for the scenarios involving SSIDs which are connected to a backend authentication server supporting EAP-NOOB are covered in Table 3. Similarly, scenarios involving SSIDs with or without EAP-NOOB support are covered in Table 4.

When every AP in the vicinity is connected to an authentication server with EAP-NOOB support, irrespective of the order of appearance of network blocks in supplicant configuration file, the method EAP-NOOB at supplicant side is called soon after starting the WPA_Supplicant. This also means that, the current protocol implementation quickly starts the authentication procedure when there are only APs connected to EAP-NOOB supporting authentication server. This argument is supported by the recorded initialization time from Table 3. Where, in each case, the EAP-NOOB method is called under three seconds. However the time necessary to display all the OOB messages grows linearly with the number of SSIDs. The in-band

SSID list	initialize time	in-band	out-of-band	Total time
EAP_NOOB_1	2.505	0.572	15.492	18.569
EAP_NOOB_1, EAP_NOOB_2	2.988	4.561	12.384	19.933
EAP_NOOB_1, EAP_NOOB_2, NOMAD	2.849	11.3295	73.0671	87.245

Table 3: Authentication latency for URL through QR-code:1

message exchange time is recorded as half a second for a single SSID and it steadily grows to reach twelve seconds for three SSIDs. The recorded out-of-band message transfer time also covers the waiting time assigned during the Waiting exchange. For anyone who is familiar with using a smart phone, scanning a QR-code is relatively a simple method for transferring OOB message. Hence, a user is expected to complete the OOB message transfer in just few seconds and the same can be seen in the first two cases in Table 3. However, in certain cases, even after the delivery of the OOB message, the supplicant may not start the Completion exchange immediately. This is because the supplicant can get assigned with an arbitrary wait time after a Waiting exchange and the SSID will be disabled in the supplicant at-least till the end of an arbitrary wait time. For the same reason, the out of band message transfer time is recorded with a higher value of seventy three seconds when authentication is attempted with three SSIDs.

Now, we will consider the scenarios involving SSIDs without the EAP-NOOB protocol support. The method initialization time depends majorly on the order of appearance of SSIDs as network blocks in the supplicant configuration file. Appearance of SSID *aalto* and *eduroam* at the beginning of configuration file will delay the EAP-NOOB initialization and in turn will the delay the authentication. This can be seen from the first two scenarios of Table 4. After the method initialization, the in-band message exchange time grows linearly with the number of SSIDs. Also as explained for Table 3, larger values for the OOB message transfer is because of an arbitrary wait time assigned during a Waiting exchange.

It can be seen that, the authentication time is more than a minute for half of the scenarios. This may affect the usability of the protocol. However, as per the current control flow of the WPA_Supplicant, aspects concerning to each SSID is handled sequentially and this increases the method initialization time and also handling an assigned wait time. Changing the control flow of the supplicant to obtain an optimum authentication latency needs complete understanding of the WPA_Supplicant architecture and also it is beyond the scope of defining a new EAP method.

SSID list	initialize time	in-band	out-of-band	Total time
EAP_NOOB_1,eduroam, aalto	30.914	0.751	19.596	51.261
EAP_NOOB_1, EAP_NOOB_2, eduroam, aalto	32.030	5.843	27.696	65.569
EAP_NOOB_1, EAP_NOOB_2, NOMAD, eduroam, aalto	9.069	10.043	58.777	77.889

Table 4: Authentication latency for URL through QR-code:2

SSID list	initialize time	in-band	out-of-band	Total time
EAP_NOOB_1 eduroam aalto	32.865	0.682	43.285	76.832

Table 5: Authentication latency for server-to-peer

5.1.2 Server-to-Peer

Similar to peer-to-server direction of authentication, the method initialization time depends on the order of occurrence of a SSID as a network block in the supplicant configuration file. Apart from the initialization, the Initial exchange with server is also decided based on the order of occurrence. Completion of the Initial exchange will allow the user to generate and deliver the QR-code. Our protocol implementation makes use of an open source package Zbarcam to read a QR-code from the web-cam. Absence of auto focus facility in the web-cam makes it challenging for a user to show the QR-code. This will increase the OOB message transfer time and will affect the total authentication time. A sample reading for server-to-peer direction can be seen in Table 5.

5.2 Use Case Scenario

A sample use case scenario where our EAP-NOOB protocol is most useful is presented here. Consider, an administrative secretary from Aalto University is supposed to install ten smart display units in Aalto University's new building. Every one of those device will connect to the network with their wireless interface. The secretary would like to establish secure connection with all the devices once they are configured. Also, he/she would like to monitor and control all the devices from a single platform.

For devices that support EAP-NOOB protocol, all that secretary has to do is to switch on each of those devices. Every incoming device authentication requests at a Wi-Fi access point will be forwarded to EAP-NOOB server which is operating from the university's cloud infrastructure. The only pending job for the secretary is to deliver the respective out-of-band message from the display unit to the web server. This is done by simply scanning a QR-code.

The secretary's Aalto user account, for example secretary@aalto.fi will be used to associate the authenticated device. The same user account will be used to fetch or deliver the OOB message during the authentication procedure. In our scenario, the secretary will scan a QR code from each of the display unit. The QR-code will decode into an URL and one click on the URL will deliver the OOB message. Once the OOB message gets delivered, the device gets authenticated automatically and now it is listed under the personnel's user account. Thereafter, all the devices can be monitored and controlled from the user account.

It should be noted that apart from delivering the out-of-band message, the secretary is not expected to do any other activity for authenticating a device and no proprietary solutions are used for completing the authentication.

6 Conclusion

In this thesis, we have addressed the security aspects of Wi-Fi connected IoT devices by implementing a secure bootstrapping method. We implemented the EAP-NOOB protocol and briefly analyzed it. The main motivation to consider the security aspect of IoT devices being, the appliances from our daily life are gaining Internet connectivity and will also be controlled remotely. Therefore, compromising the security of those devices can cause harm to the users. The primary objective of this thesis was to build a prototype of EAP-NOOB protocol for authenticating and connecting IoT devices to an access network. Additionally, the authenticated device is associated with a user account to enable remote monitoring and supervision.

In this thesis, we first presented the problem statement and research goal in chapter 1. Next in chapter 2, we introduced all the technologies and topics necessary to understand both the IoT device security problem and the proposed solution, EAP-NOOB. The complete description of the EAP-NOOB protocol, including the security was presented as part of chapter 3. The thesis project focused on implementing the EAP-NOOB protocol. We discussed the development environment, source packages and applications developed for implementing the protocol in chapter 4. Finally, we did an authentication latency analysis over the implemented EAP-NOOB prototype in chapter 5.

In this thesis, we have successfully implemented a prototype to authenticate devices in both directions. Also, the prototype currently supports two kinds of OOB message passing: QR-code and NFC. The prototype was able to confirm the feasibility of deploying the formulated EAP-NOOB protocol over a real world network. The implemented prototype was tested on a sample smart device (Laptop). Using EAP-NOOB as the EAP method, the device was able to authenticate and associate with a user account in the online server and was also able to join a secure Wi-Fi network. Thus, it can be said that, this thesis has successfully implemented and tested the EAP-NOOB protocol. Additionally, defining the EAP-NOOB protocol is an ongoing process. The protocol specification and its implementation are done by our research group. The EAP-NOOB prototype developed in this thesis has helped in specifying the protocol as it provided means to test and analyse the protocol. Currently, the protocol is undergoing the standardisation process with the IETF and developing the EAP-NOOB prototype has directly contributed to improve the protocol.

The usability of new technology makes a big difference to its adoption. Completing the authentication with the current implementation only requires performing familiar tasks such as scanning a QR-code or tapping a tag to read through NFC. To assess the ease of usage, the prototype usability study should still be undertaken. The authentication time analysis from chapter 5 shows that the time necessary for the completion of the authentication in certain scenarios is in the order of minutes. For such scenarios, feedback from users should be collected to see if waiting a few minutes would affect the deployment of the protocol. The protocol specification considers many most known attacks and security vulnerabilities. However, it is still possible to have unforeseen vulnerabilities, which may be exploited after the deployment of the protocol. Therefore, a formal analysis on EAP-NOOB would help to check the robustness of the protocol and to highlight any weaklinks in it.

As the processors and memory are getting cheaper, appliances and devices from our daily-life are becoming more intelligent. Smart home solutions from major home appliance vendors are already on the market. With time, these solutions will make their way to our household and work environment. Everyday life from our future is expected to be lot more convenient with these smart solutions around. However, it is not necessary to wait for an unfortunate security incident to realize the degree of control and the level of closeness the devices have over our daily life. As a famous saying suggests, *"Prevention is better than cure"*. It will be wise to investigate and be ready with security solutions which can make our home and work environment secure. In this thesis, we have implemented and analyzed one such solution, which aims to secures the Internet connect IoT devices through secure bootstrapping.

References

[1] In Lee and Kyoochun Lee. "The Internet of Things (IoT): Applications, investments, and challenges for enterprises", in *Business Horizons* vol. 58, is. 4, pp. 431–440, 2015.

[2] Umesh Hodeghatta Rao and Umesha Nayak, "Current Trends in Information Security", in*The InfoSec Handbook* pp. 325-330, 2014.

[3] Karolis Vilius, Lu Liu, John Panneerselvam and Thomas Stimpson, "A Critical Analysis of the Efficiencies of Emerging Wireless Security Standards Against Network Attacks", in *Intelligent Networking and Collaborative Systems*, Taipei, Taiwan, Sep. 2-4, 2015.

[4] Dave Evans, "The Internet of Things How the Next Evolution of the Internet Is Changing Everything", Cisco white paper, Apr. 2011.

[5] Haijun Zhang, Xiaoli Chu, Weisi Guo and Siyi Wang, "Coexistence of Wi-Fi and heterogeneous small cell networks sharing unlicensed spectrum", *IEEE Communications Magazine*, vol. 53, is. 3, pp 158-164, Mar. 2015.

[6] Ji-chun Zhao, Jun-feng Zhang, Yu Feng and Jian-xin Guo, "The study and application of the IOT technology in agriculture", In *IEEE ICCSIT 2010*, 9-11 Jul. 2010.

[7] Andrea Zanella, Nicola Bui, Angelo Castellani, Lorenzo Vangelista and Michele Zorzi, "Internet of Things for Smart Cities", In *IEEE Internet of Things Journal*, vol. 1, is. 1, pp 22-32, Feb. 2014.

[8] P.S. Henry and Hui Luo, "WiFi: what's next?", in *IEEE Communications Magazine*, vol. 40, is. 12, Dec. 2002.

[9] *IEEE Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications*, IEEE 802.11-1997,1997.

[10] Martin Beck and Erik Tews, *Practical Attacks Against WEP and WPA*, Wireless network security 09, Zurich, Switzerland, Mar. 2009.

[11] Vipul Gupta, Sumit Gupta Sheueling Chang and Douglas Stebila , "Performance Analysis of Elliptic Curve Cryptography for SSL", in *Proceedings of the 1st ACM workshop on Wireless security*, pp. 87-94, Atlanta, GA, USA, Sep. 2002.

[12] Nancy Cam-Winget, Russ Housley, David Wagner and Jesse Walker, "Security flaws in 802.11 data link protocols", in *Communication of ACM - Wireless Networking Security*, vol. 46 is. 5, pp. 35-39 May 2003

[13] Kenneth G. Paterson, Bertram Poettering, Jacob C. N. Schuld, "Plaintext Recovery Attacks Against WPA/TKIP", in *Fast Software Encryption*, Springer Link, vol. 8540, pp 325-349, 19 Apr. 2015.

[14] Avishai Wool, "A Note on the Fragility of the "Michael" Message Integrity Code", in *IEEE Transaction On Wireless Communications*, vol. 3, is. 5, Sep. 2004.

[15] Gunther Lackner, "A Comparison of Security in Wireless Network Standards with a Focus on Bluetooth, WiFi and WiMAX ", in *International Journal of Network Security*, vol. 15, no. 6, pp. 420-436, Nov. 2013.

[16] Dimitris Zisiadis, Spyros Kopsidas, Argyris Varalis and Leandros Tassiulas, "Enhancing WPS security", in *Wireless Days (WD), 2012 IFIP*, Ireland, Nov. 21-23, 2012.

[17] Amirali Sanatinia, Sashank Narain and Guevara Noubir, "Wireless spreading of WiFi APs infections using WPS flaws: An epidemiological and experimental study" , in *IEEE Conf. Communications and Network Security 2013*, Washington, DC, USA, 14-16 Oct. 2013.

[18] *Extensible Authentication Protocol (EAP)*, RFC3748, IETF, Jun. 2014.

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

[20] Khidir M. Ali and Ali Al-Khlifa, "A Comparative Study of Authentication Methods for Wi-Fi Networks", in *IEEE Computational Intelligence, Communication Systems and Networks*, Bali, Indonesia, 26-28 Jul. 2011.

[21] *Internet X.509 Public Key Infrastructure Certificate and CRL Profile*, RFC2459, IETF, Jan. 1999.

[22] Sapna Sejwani and Sarvesh Tanwar, "Implementation of X.509 Certificate for Online Applications ", in *International Journal of Research in Advent Technology*, Vol.2, No.3, PP. 250-254 Mar. 2014.

[23] *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, RFC3280, IETF, Apr. 2002.

[24] Moeen Hassanalieragh, Alex Page, Tolga Soyata, Gaurav Sharma, Mehmet Aktas, Gonzalo Mateos, Burak Kantarci and Silvana Andreescu, "Health Monitoring and Management Using Internet-of-Things (IoT) Sensing with Cloud-Based Processing: Opportunities and Challenges", *2015 IEEE International Conference on Services Computing (SCC)*, San Fransisco USA, Jun. 2015.

[25] Roberto Minerva, Abyi Biru and Domenico Rotondi, "Towards a definition of the Internet of Things (IoT)", IEEE Internet Initiative, Rev. 1, 27 May 2015.

[26] Rodrigo Roman ,Pablo Najera and Javier Lopez,"Securing the Internet of Things", in *Computer*, vol. 44, is. 9, pp. 51-58 Sept. 2011.

[27] Qi Jing, Athanasios V. VasilakosJiafu Wan and Jingwei LuDechao Qiu, "Security of the Internet of Things: perspectives and challenges", in *Wireless Networks*, Springer Link, Vol. 20, Is. 8, pp 2481–2501, Nov. 2014.

[28] Tuomas Aura and Dieter Gollmann, "Communications security on the internet", in *Software Focus*, vol. 2, is. 3, pp. 104-111, Sep. 2001.

[29] Antonio J. Jara Valera, Miguel A. Zamora, and Antonio F. G. Skarmeta, "An Architecture Based on Internet of Things to Support Mobility and Security in Medical Environments", *IEEE 7th Consumer Communications and Networking Conference (CCNC)*, Las Vegas, Nevada USA, Jan 2010.

[30] Definition of bootstrap in English, "bootstrap", Internet: `https://en.oxforddictionaries.com/definition/bootstrap` [Nov. 4 2016]

[31] *Wi-Fi Device Provisioning Protocol (DPP) Technical Specification*, version 0.0.23, 2016.

[32] *Datagram Transport Layer Security Version 1.2*, RFC6347, IETF, 2012.

[33] Liang Zhou and Han-Chieh Chao, "Multimedia traffic security architecture for the internet of things", in *IEEE Network*, vol. 25, is. 3, 23 May 2011.

[34] Sanaz Rahimi Moosavi, Tuan Nguyen Gia, Amir-Mohammad Rahmani, Ethiopia Nigussie, Seppo Virtanen, Jouni Isoaho and Hannu Tenhunen, "SEA: A Secure and Efficient Authentication and Authorization Architecture for IoT-Based Healthcare Using Smart Gateways", in *Procedia Computer Science*, vol. 52, pp. 452-459, 2015.

[35] Behcet Sarikaya and Mohit Sethi, "Secure IoT Bootstrapping: A Surve draft-sarikaya-t2trg-sbootstrapping-01",
Internet: `https://tools.ietf.org/html/draft-sarikaya-t2trg-sbootstrapping-01`, Jul. 1 2016 [Nov. 11 2016].

[36] Shahab Mirzadeh, Haitham Cruickshank and Rahim Tafazolli, "Secure Device Pairing: A Survey", *IEEE Communications Survays and Tutorials*, vol. 16, no. 1, Q1 2014.

[37] Whitfiled Diffie and Martin E Hellman, "New Directions in Cryptography", *IEEE Transaction on Information Theory*, vol 22, no. 6, Nov 1976.

[38] Kristin Lauter, "The Advantages Of Elliptic Curve Cryptography For Wireless Security", in *IEEE Wireless Communications*, pp. 62-67, Feb. 2004.

[39] Li Li, Hu Xiaoguang, Chen Ke and He Ketai, "The applications of WiFi-based Wireless Sensor Network in Internet of Things and Smart Grid", *IEEE Conference on Industrial Electronics and Applications*, Beijing, China, 21-23 June 2011.

[40] *Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography*, NIST Special Publication 800-56A, Mar. 2007.

[41] *State Machines for Extensible Authentication Protocol (EAP) peer and authenticator*, RFC4137, IETF, Aug. 2007.

[42] Tuomas Aura and Mohit Sethi, "Nimble out-of-band authentication for EAP (EAP-NOOB) draft-aura-eap-noob-01", Internet: `https://datatracker.ietf.org/doc/draft-aura-eap-noob/?include_text=1`, Jul. 2016 [Nov. 29 2016]

[43] Weiping Sun, Munhwan Choi and Sunghyun Choi, "IEEE 802.11ah: A Long Range 802.11 WLAN at Sub 1 GHz", Internet: *IEEE 802.11ah: A Long Range 802.11 WLAN at Sub 1 GHz*, May. 2013 [Jan. 6 2017]

[44] *Diameter Base Protocol*, RFC6733, IETF, Oct. 2012.

[45] Xing Zhang, Shaohua Ma, Dong Han and Wei Shi,"Implementation of elliptic curve Diffie-Hellman key agreement scheme on IRIS nodes", *International Conference on Intelligent Computing and Internet of Things*, Harbin, China, Jan. 2015.

[46] Kai Lei, Yining Ma and Zhi Tan, "Performance Comparison and Evaluation of Web Development Technologies in PHP, Python, and Node.js", *2014 IEEE 17th International Conference on Computational Science and Engineering*, Chengdu, China, Dec. 2014.

[47] Smart shirt, "Ralph Lauren Launches Wearable IoT Smart Shirt", `http://www.machinetomachinemagazine.com/2015/08/29/ralph-lauren-launches-wearable-iot-smart-shirt/` [12.01.2017]

[48] Mohit Sethi, Pranvera Kortoci, Mario Di Francesco, and Tuomas Aura, "Secure and low-power authentication for resource-constrained devices", *5th International Conference on the Internet of Things (IOT), IEEE*, Seoul, South Korea, Oct. 2015 .

[49] Mohit Sethi, Elena Oat, Mario Di Francesco, and Tuomas Aura, "Secure bootstrapping of cloud-managed ubiquitous displays", *International Joint Conference on Pervasive and Ubiquitous Computing*, ACM, Seattle, Washington, pp. 739-750, Sep. 2014.

A EAP-NOOB Parameters

Vers, Verp	Supported protocol versions. Vers is a JSON array of unsigned integers and Verp is an unsigned integer. The current supported values are [1] and 1.
PeerID	Peer identifier assigned by the server. The identifier should be utf8 characters and maximum 60 characters. Until Completion exchange, the assigned identifier is ephemeral. One way to generate the identifier is by generating upto 60 digit random lower-case hexadecimal string.
Type	EAP-NOOB message type ranging from 0-7. The parameter is of integer type.
PKp, PKs	peer and server public component of the generated ECDH key. Parameters are sent in JSON webkey format
Cryptosuites, Cryptosuitep	Supported peer and server cryptographic algorithms and ECDH curve.
Dirs, Dirp	Supported peer and server directions. The possible values for the parameters are: 1=peer-to-server, 2=server-to-peer and 3=both. The parameters are of unsigned integer type.
Ns, Np	peer and server nonces for Initial exchange. The parameter should be of 16 byte length and should be encoded in base64url format.
ServerInfo	Information particular to EAP-NOOB server and web server encoded as a JSON object. A user identifies the server based on the information shared as part of this parameter. The total length must not exceed 500 bytes
PeerInfo	similar to ServerInfo, the parameter carries information particular to the client device. A user can identify the peer device at the web server portal using the information shared as this parameter. The total length must not exceed 500 bytes
Minsleep	Wait time in minutes assigned by server during Waiting exchange. Parameter is of unsigned integer type and the value ranges from 0..3600.
noob	a 16 byte nonce value for OOB message. The parameter should be encoded in base64url format.
hoob	A 16 byte cryptographic fingerprint calculated using Initial exchange parameters. The parameter should be encoded in base64url format.
Np2, Ns2	peer and server nonces for Reconnect exchange. The parameter should be of 16 byte length and should be encoded in base64url format.
MACp, MACs	peer and server message authentication code for Completion exchange. The parameter should be of 16 byte length and should be encoded in base64url format.

PKp2, PKs2	peer and server public component of the generated ECDH key. The parameters are exchanged only if a new cryptosuite is negotiated. Parameters are sent in JSON webkey format
MACp2, MACs2	peer and server message authentication code for Reconnect exchange. The parameter should be of 16 byte length and should be encoded in base64url format.

Table 6: EAP-NOOB parameters